
LSF Batch User's Guide

Sixth Edition, August 1998

Platform Computing Corporation

LSF Batch User's Guide

Copyright © 1994-1998 Platform Computing Corporation
All rights reserved.

This document is copyrighted. This document may not, in whole or part, be copied, duplicated, reproduced, translated, electronically stored, or reduced to machine readable form without prior written consent from Platform Computing Corporation.

Although the material contained herein has been carefully reviewed, Platform Computing Corporation does not warrant it to be free of errors or omissions. Platform Computing Corporation reserves the right to make corrections, updates, revisions or changes to the information contained herein.

UNLESS PROVIDED OTHERWISE IN WRITING BY PLATFORM COMPUTING CORPORATION, THE PROGRAM DESCRIBED HEREIN IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL PLATFORM BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, ARISING OUT OF THE USE OR INABILITY TO USE THIS PROGRAM.

LSF Base, LSF Batch, LSF JobScheduler, LSF MultiCluster, LSF Analyzer, LSF Make, LSF Parallel, Platform Computing, and the Platform Computing and LSF logos are trademarks of Platform Computing Corporation.

Other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations.

Printed in Canada

Revision Information for LSF Batch User's Guide

Edition	Description
First	This document describes LSF 2.0. Based on Chapter 1, Introduction to LSF, from the LSF User's and Administrator's Guide, second edition.
Second	Revised to incorporate the LSF 2.1 Release Notes.
Third	Revised to reflect the changes in LSF 2.2.
Fourth	Revised to describe the new features of LSF Suite 3.0.
Fifth	Revised to describe the new features of LSF Suite 3.1.
Sixth	Revised to describe the new features of LSF Suite 3.2.

Contents

Preface	xiii
Audience	xiii
LSF Suite 3.2	xiii
LSF Enterprise Edition	xiv
LSF Standard Edition	xiv
Related Documents	xiv
Online Documentation	xv
Technical Assistance	xv
 1 - Introduction	 1
What is LSF?	1
LSF Features	1
Host Resources	3
Batch Processing	4
Interactive Processing	4
Clusters	4
Fault Tolerance	5
Structure of LSF Base	6
Load Information Manager	6
Remote Execution Server	7
LSF API	7
LSF Utility Programs	7
Applications	9
Structure of LSF Batch	10
 2 - Getting Started	 13
Getting Cluster Information	13
Displaying the Cluster and Master Names	13
Displaying Available Resources	14
Getting Host Information	16
Displaying Static Host Information	16
Displaying Load Information	17

Contents

Running Jobs.	18
Running Jobs on Remote Hosts	18
Load Sharing Commands With <code>lscsh</code>	19
Parallel Processing With LSF Make	19
Listing Hosts	20
Submitting a Job	20
Selecting a Job Queue.	21
Tracking Batch Jobs	22
<code>xbsub</code> and <code>xlSBATCH</code> GUI Applications	23
3 - Cluster Information	25
Finding the Master.	25
Listing Resources	26
Listing Hosts.	27
Displaying the Load	29
Graphical Load Display	30
4 - Resources	35
Introduction to Resources.	35
Load Indices.	37
Static Resources.	41
Shared Resources	43
Boolean Resources	45
Listing Resources	46
Resource Requirement Strings.	46
Selection String	47
Order String.	49
Resource Usage String	50
Job Spanning String	51
Specifying Shared Resources	52
Configuring Resource Requirements	53
Remote Task List File	53
Managing Your Task List	53
Using Resource Requirements.	54
5 - Using LSF Batch	55
Batch Jobs	56
Fairshare Scheduling Policy	58
Host Partition Fairshare Scheduling	59
Queue-Level Fairshare Scheduling	59

Hierarchical Fairshare	60
Other Scheduling Policies	62
Preemptive Scheduling	62
Exclusive Scheduling	63
Processor Reservation	63
Backfill Scheduling	64
Scheduling Parameters	64
Load Thresholds	64
Scheduling Conditions	65
Time Windows for Queues and Hosts	66
Run Windows	66
Dispatch Windows	66
Batch Queues	67
Finding Out What Queues Are Available	67
Detailed Queue Information	69
Automatic Queue Selection	75
Specifying Default Queues	76
Queue Selection Mechanism	76
Choosing a Queue	77
Batch Users	78
Batch Hosts	79
User and Host Groups	81
Viewing Hierarchical Share Information	82
Queue-Level Job Starters	84
Configuration Parameters	85
User Controlled Account Mapping	86
6 - Submitting Batch Jobs	89
Input and Output	90
Resource Requirements	91
Resource Reservation	91
Host Selection	93
Host Preference	94
Resource Limits	95
Pre-Execution Commands	97
Job Dependencies	98
Job Dependency Examples	100
Remote File Access	101
Start and Termination Time	103
Parallel Jobs	103

Contents

Minimum and Maximum Number of Processors	104
Specifying Locality	104
Job Arrays	106
Creating a Job Array	107
LSB_JOBINDEX Environment Variable	109
Array Job Dependencies	109
Handling Input/Output/Error Files for Job Arrays	110
Specifying a Share Account	111
Re-initializing Job Environment on the Execution Host	111
Other bsub Options	112
Job Scripts	114
Embedded Submission Options	115
Running a Job Under a Particular Shell	116
Submitting Jobs Using the Job Submission GUI	117
7 - Tracking Batch Jobs	119
Displaying Job Status	119
Finding Pending or Suspension Reasons	120
Monitoring Resource Consumption of Jobs	122
Displaying Job History	123
Viewing Chronological History	125
Checking Partial Job Output	126
Tracking Job Arrays	126
Displaying Queue and Host Status	127
Job Controls	127
Killing Jobs	128
Suspending and Resuming Jobs	128
Controlling Job Arrays	129
Sending an Arbitrary Signal to a Job	130
Moving Jobs Within and Between Queues	131
Job Modification	132
Submitted Job Modification	132
Dispatched Job Modification	133
Job Array Modification	133
Job Tracking and Manipulation Using the GUI	135
8 - Running Interactive Jobs	139
Shared Files and User IDs	139
Running Remote Jobs with <code>lsrun</code>	140
Running Parallel Jobs with <code>lsgrun</code>	141

Load Sharing Interactive Sessions	142
Load Sharing Login	142
Load Sharing X Sessions	142
Command-Level Job Starters	144
Interactive Batch Job Support	145
Shell Mode for Remote Execution	147
9 - Using <code>lscsh</code>	149
Starting <code>lscsh</code>	149
Using <code>lscsh</code> as Your Login Shell	150
Automatic Remote Execution	150
Host Redirection	151
Job Control	152
Built-in Commands	152
The <code>lsmode</code> Command	153
The <code>connect</code> Command	154
Modes of Operation	154
Differences from Other Shells	155
Writing Shell Scripts in <code>lscsh</code>	156
Limitations	156
10 - Using LSF Make	159
Parallel Execution	159
Invoking LSF Make	160
Specifying the Number of Processors	160
File Server Load	160
Tuning Your Makefile	161
Building in Subdirectories	161
Running <code>lsmake</code> as a Batch Job	162
Differences from Other Versions of <code>make</code>	163
11 - Checkpointing and Migration	165
Approaches to Checkpointing	166
Kernel-level Checkpointing	166
User-level Checkpointing	166
Application-level Checkpointing	166
Checkpoint Directory	167
Uniform Checkpointing Interface	167
The <code>echkpnt</code> Command	167
The <code>erestart</code> Command	168

Contents

Submitting Checkpointable Jobs	169
Checkpointing a Job.	170
Restarting a Checkpointed Job.	171
Job Migration	173
Queues and Hosts for Automatic Job Migration	174
Automatically Rerunning and Restarting Jobs	174
Submitting a Job for Automatic Migration	174
Building Checkpointable Jobs	175
The Checkpoint Library	175
The Checkpoint Startup Routine	175
Linking	175
Limitations	178
12 - Customizing Batch Jobs for LSF	179
Environment Variables	179
Parallel Jobs.	181
Getting the Host List	181
Starting Parallel Tasks With <code>lstools</code>	182
Using LSF Make to Run Parallel Batch Jobs	182
Submitting PVM Jobs to LSF Batch	183
Submitting MPI Jobs to LSF Batch	183
Submitting POE Jobs to LSF Batch	185
Using a Job Starter for Parallel Jobs	186
13 - Using LSF MultiCluster	187
What is LSF MultiCluster?	187
Getting Remote Cluster Information	188
Running Batch Jobs across Clusters	189
Running Interactive Jobs on Remote Clusters	192
User-Level Account Mapping Between Clusters	192
14 - Interoperation with NQS	195
Choosing an LSF Batch Queue	196
Submitting a Job from LSF to NQS	196
Controlling Jobs Running on NQS	197
Forwarding of Output Files	197

A - Customizing <code>xlsbatch</code> Menu Items.	199
B - Frame Arrays.	203
Overview.	203
Distribution.	204
Frame Array Concepts	204
Submitting Frame Arrays	205
Tracking Frame Arrays	208
Controlling Frame Arrays.	208
C - Using LSF with Alias Renderer	211
Overview.	211
Distribution.	212
Installing the queue-level job starter.	212
Submitting Checkpointed Frame Arrays	212
Tracking Checkpointed Frame Arrays	214
D - Using LSF with FLUENT	215
Overview.	215
Distribution.	216
Configuring the Checkpointing Executable Files	216
Submitting the FLUENT Job	217
Checkpointing the FLUENT job	217
Restarting the FLUENT job	218
Index.	219

Contents

Preface

Audience

This guide provides tutorial and reference information for users of LSF Base, LSF Batch and LSF MultiCluster. Users should be familiar with executing commands in a UNIX or Windows NT environment.

LSF Batch administrators should also be familiar with the contents of this guide, as well as those of the *LSF Batch Administrator's Guide*.

LSF Suite 3.2

LSF is a suite of workload management products including the following:

LSF Batch is a batch job processing system for distributed and heterogeneous environments, which ensures optimal resource sharing.

LSF JobScheduler is a distributed production job scheduler that integrates heterogeneous servers into a virtual mainframe or virtual supercomputer

LSF MultiCluster supports resource sharing among multiple clusters of computers using LSF products, while maintaining resource ownership and cluster autonomy.

LSF Analyzer is a graphical tool for comprehensive workload data analysis. It processes cluster-wide job logs from LSF Batch and LSF JobScheduler to produce statistical reports on the usage of system resources by users on different hosts through various queues.

Preface

LSF Parallel is a software product that manages parallel job execution in a production networked environment.

LSF Make is a distributed and parallel Make based on GNU Make that simultaneously dispatches tasks to multiple hosts.

LSF Base is the software upon which all the other LSF products are based. It includes the network servers (LIM and RES), the LSF API, and load sharing tools.

There are two editions of the LSF Suite:

LSF Enterprise Edition

Platform's LSF Enterprise Edition provides a reliable, scalable means for organizations to schedule, analyze, and monitor their distributed workloads across heterogeneous UNIX and Windows NT computing environments. LSF Enterprise Edition includes all the features in LSF Standard Edition (LSF Base and LSF Batch), plus the benefits of LSF Analyzer and LSF MultiCluster.

LSF Standard Edition

The foundation for all LSF products, Platform's Standard Edition consists of two products, LSF Base and LSF Batch. LSF Standard Edition offers users robust load sharing and sophisticated batch scheduling across distributed UNIX and Windows NT computing environments.

Related Documents

The following guides are available from Platform Computing Corporation:

- LSF Installation Guide*
- LSF Batch Administrator's Guide*
- LSF Batch Administrator's Quick Reference*
- LSF Batch User's Guide*
- LSF Batch User's Quick Reference*
- LSF JobScheduler Administrator's Guide*
- LSF JobScheduler User's Guide*

LSF Analyzer User's Guide
LSF Parallel User's Guide
LSF Programmer's Guide

Online Documentation

- Man pages (accessed with the `man` command) for all commands
- Online help available through the Help menu for the `xlsbatch`, `xbmod`, `xbsub`, `xbalarms`, `xbcal` and `xlsadmin` applications.

Technical Assistance

If you need any technical assistance with LSF, please contact your reseller or Platform Computing's Technical Support Department at the following address:

LSF Technical Support
Platform Computing Corporation
3760 14th Avenue
Markham, Ontario
Canada L3R 3T7

Tel: +1 905 948 8448
Toll-free: 1-87PLATFORM (1-877-528-3676)
Fax: +1 905 948 9975
Electronic mail: *support@platform.com*

Please include the full name of your company.

You may find the answers you need from Platform Computing Corporation's home page on the World Wide Web. Point your browser to *www.platform.com*.

If you have any comments about this document, please send them to the attention of LSF Documentation at the address above, or send email to *doc@platform.com*.

1. Introduction

This section of the LSF Batch User's Guide gives a quick introduction to LSF Base, LSF Batch, LSF Make and LSF MultiCluster products. After reading the conceptual material, you should be able to begin using LSF. The rest of the guide contains more detailed information on LSF features and commands.

What is LSF?

LSF is a suite of workload management products from Platform Computing Corporation. The LSF Suite includes LSF Batch, LSF JobScheduler, LSF MultiCluster, LSF Make, and LSF Analyzer all running on top of the LSF Base system.

LSF manages, monitors, and analyzes the workload for a heterogeneous network of computers and it unites a group of UNIX and NT computers into a single system to make better use of the resources on a network. Hosts from various vendors can be integrated into a seamless system. You can submit your job and leave the system to find the best host to run your programs.

LSF supports sequential and parallel applications running as interactive and batch jobs. LSF also allows new distributed applications to be developed through LSF Application Programming Interface (API), C programming libraries and a tool kit of programs for writing shell scripts.

LSF Features

With LSF you can use a network of heterogeneous computers as a single system. You are no longer limited to the resources on your own workstation. You do not need to rewrite or change your programs to take advantage of LSF. You only need to learn a few simple commands and the resources of your entire network will be within reach.

1 Introduction

LSF can automatically select hosts in a heterogeneous environment based on the current load conditions and the resource requirements of the applications.

With LSF, remotely run jobs behave just like jobs run on the local host. Even jobs with complicated terminal controls behave transparently to the user as if they were run locally.

LSF can run batch jobs automatically when required resources become available, or when systems are lightly loaded. LSF maintains full control over the jobs, including the ability to suspend and resume the jobs based on load conditions.

LSF can run both sequential and parallel jobs. Some jobs speed up substantially when run on a group of idle or lightly loaded hosts. For example, the LSF Make program allows you to do your software builds or automated tests many times faster than with traditional makes.

With LSF, you can transparently run software that is not available on your local host. For example, you could run a CAD tool that is only available on an HP host from your Sun workstation. The job would run on the HP and be displayed transparently on your Sun system.

With LSF, the system administrators can easily control access to resources such as:

- who can submit jobs and which hosts they can use
- how many jobs specific users or user groups can run simultaneously
- time windows during which each host can run load shared jobs
- load conditions under which specific hosts can accept jobs or have to suspend some jobs
- resource limits for jobs submitted to specific queues

LSF provides mechanisms for resource and job accounting. This information can help the administrator to find out which applications or users are consuming resources, at what times of the day (or week) the system is busy or idle, and whether certain resources are overloaded and need to be expanded or upgraded.

LSF allows you to write your own load sharing applications, both as shell scripts using the `lstoold`s programs and as compiled programs using the LSF application programming libraries.

Host Resources

LSF provides comprehensive resource and load information about all hosts in the network.

Resource information includes the number of processors on each host, total physical memory available to user jobs, the type, model, and relative speed of each host, the special resources available on each host, and the time windows when a host is available for load sharing.

Dynamic load information includes:

- CPU load
- available real memory
- available swap space
- paging activity
- I/O activity
- number of interactive users logged in
- interactive idle time
- space in the `/tmp` directory
- arbitrary site-defined load indices

Batch Processing

LSF Batch lets you submit *batch* jobs to a *queue*, which can have access to many hosts on your network, and can automatically run jobs as soon as a suitable host is available. Resource intensive jobs are processed more efficiently because they are scheduled automatically. You do not have to spend time hunting around on the network to find an idle host with the resources that your job needs.

The system administrator can create multiple queues, and can specify policies for each queue that will help to prioritize and schedule the work.

Interactive Processing

LSF lets you run *interactive* jobs on any computer on the network, using your own terminal or workstation. Interactive jobs run immediately and normally require some input through a text-based or graphical user interface. All the input and output is transparently sent between the local host and the job execution host.

You can submit interactive jobs using LSF Batch to take advantage of queues and queuing policies. However, an interactive batch job is subject to the scheduling policies of the submission queue, so it may not be dispatched immediately.

Clusters

Load sharing in LSF is based on *clusters*. A cluster is simply a group of hosts. Each cluster has one or more LSF administrators. An administrator is a user account that has permission to change the LSF configuration and perform other maintenance functions. An LSF administrator decides how the hosts are grouped together.

A cluster can contain a mixture of host types. By putting all hosts types into a single cluster, you can have easy access to the resources available on all host types.

Clusters are normally set up based on administrative boundaries. LSF clusters work best when each user has an account on all hosts in the cluster, and user files are shared among the hosts so that they can be accessed from any host. This way LSF can send a job to any host. You need not worry about whether the job will be able to access the correct files.

LSF can also run batch jobs when files are not shared among the hosts. LSF includes facilities to copy files to and from the host where the batch job is run, so your data will always be in the right place.

LSF can also run batch jobs when user accounts are not shared by all hosts in a cluster. Accounts can be mapped across machines.

LSF MultiCluster supports interoperation across clusters. Your jobs can be forwarded transparently to be run on another cluster within your organization.

Fault Tolerance

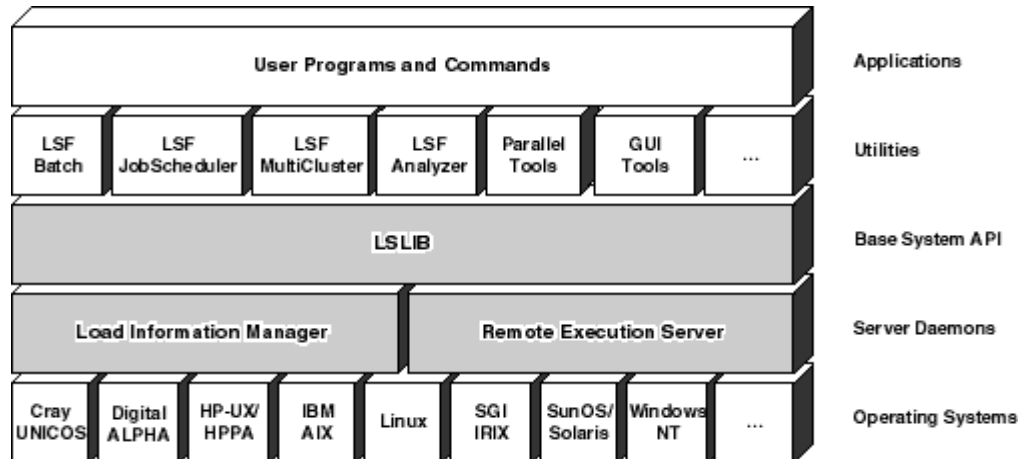
LSF is designed to continue operating even if some of the hosts in the cluster are unavailable. One host in the cluster acts as the master, but if the master host becomes unavailable another host takes over. LSF services are available as long as there is one available host in the cluster.

When a host crashes, all jobs running on that host are lost. No other pending or running jobs are affected. Important jobs can be submitted to LSF Batch with an option to automatically restart if the job is lost because of a host failure.

Structure of LSF Base

Figure 1 shows the structure of LSF Base and how it fits into your system. The software modules that make up LSF Base are shaded.

Figure 1. Structure of LSF Base



A server is a host that runs load shared jobs. The *Load Information Manager* (LIM) and *Remote Execution Server* (RES) run on every server. The LIM and RES are implemented as daemons that interface directly with the underlying operating systems and provide users with a uniform, host independent environment. The *Load Sharing LIBrary* (LSLIB) is the basic interface.

The LIM, RES and LSLIB form the LSF Base system.

Load Information Manager

The LIM on each server monitors its host's load and exchanges load information with other LIMs. On one host in the cluster, the LIM acts as the master. The master LIM collects information for all hosts and provides that information to the applications. The master LIM is chosen among all the LIMs running in the cluster. If the master LIM

becomes unavailable, a LIM on another host will automatically take over the role of master.

The LIM provides simple placement advice for interactive tasks. This information is used by some of the `lstoools(1)` applications (for example, `lshrun`) to determine which host to run on.

Remote Execution Server

The RES on each server accepts remote execution requests and provides fast, transparent and secure remote execution of interactive jobs.

LSF API

LSLIB is the *Application Programming Interface* (API) for the LSF Base system, providing easy access to the services of LIM and RES.

LSF Utility Programs

The LSF utilities are a suite of products built on top of LSF Base. The utilities include the following products:

LSF Batch

The LSF Batch queuing system uses dynamic load information from the LIM to schedule batch jobs in an LSF cluster. LSF Batch is described further in the section ‘Structure of LSF Batch’ on page 10.

LSF JobScheduler

The LSF JobScheduler is a separately licensed product of LSF that manages data processing workload in a distributed environment.

LSF MultiCluster

A very large organization may divide its computing resources into a number of autonomous clusters, reflecting the structure of the company. The separately licensed LSF MultiCluster product enables load sharing across clusters resulting in more

1 Introduction

efficient use of the resources of the entire organization. LSF MultiCluster is described in more details in ‘Using LSF MultiCluster’ on page 187.

LSF Analyzer

LSF Analyzer is a graphical tool for comprehensive workload data analysis. It processes cluster-wide job logs from LSF Batch and LSF JobScheduler to produce statistical reports on the usage of system resources by users on different hosts through various queues.

Command Interpreter

`lstcsh` is a load-sharing version of `tcsh`, a popular UNIX command interpreter (shell). In addition to all the features of `tcsh`, `lstcsh` allows arbitrary UNIX commands and user programs to be executed remotely. Remote execution with `lstcsh` is completely transparent. For example, you can run `vi` remotely, suspend it, and resume it. For more information, see ‘Using `lstcsh`’ on page 149.

LSF Make

LSF Make is a load sharing version of GNU make. It uses the same makefiles as GNU make and behaves similarly, except that you specify the number of hosts to use to run the make tasks in parallel. Tasks are started on multiple hosts simultaneously to reduce the execution time.

`lsmake`, the LSF Make executable, is covered by the Free Software Foundation General Public License. `tcsh` is covered by copyrights held by the University of California. Read the file `LSF_MISC/lsmake/COPYING` in the LSF software distribution for details.

Load Sharing Tools

The `lstools` are a set of utilities for getting information from LSF and running programs on remote hosts. For example, you can write a script that uses the `lstools` to find the best hosts satisfying given resource requirements, then run jobs on one or more of the selected hosts.

Parallel Tools

The parallel tools are a set of utilities for users to run parallel applications using message passing packages. PVM and MPI jobs can be submitted to the LSF Batch system through `pvmjob` and `mpijob`, shell scripts for running PVM and MPI jobs under LSF Batch. See ‘Parallel Jobs’ on page 181 for more information.

GUI Tools

LSF has a comprehensive set of Graphical User Interface tools that give users complete access to the power and flexibility of the LSF Suite with the convenience of point and click.

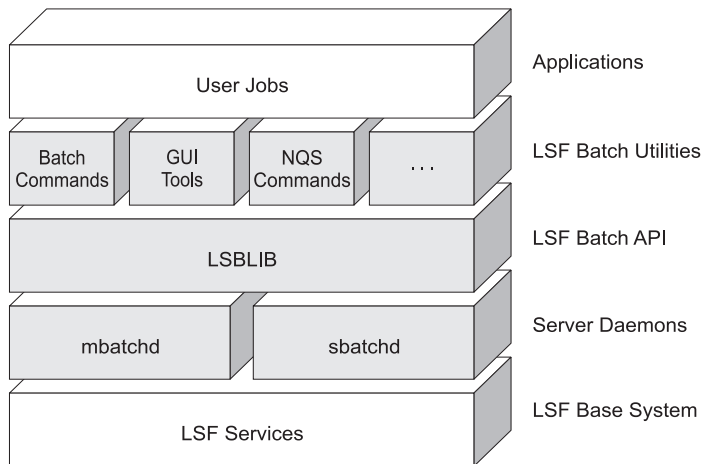
Applications

Most applications can access LSF through LSF utility programs. Most applications do not need to communicate directly with LSF and do not need to be modified to work with LSF. Nearly all UNIX or Windows NT commands and third-party applications can be run using LSF utilities.

Structure of LSF Batch

LSF Batch is a distributed batch system built on top of the LSF Base system to provide powerful batch job scheduling services to users. *Figure 2* shows the components of LSF Batch and the interactions among them.

Figure 2. Structure of LSF Batch



LSF Batch accepts user jobs and holds them in queues until suitable hosts are available. Host selection is based on up-to-date load information from the master LIM, so LSF Batch can take full advantage of all your hosts without overloading any.

LSF Batch runs user jobs on suitable server hosts that the LSF administrator has chosen. LSF Batch has sophisticated controls for sharing hosts with interactive users, so you do not need dedicated hosts to process batch jobs.

There is one master batch daemon (`mbatchd`) running in each LSF cluster, and one slave batch daemon (`sbatchd`) running on each batch server host. User jobs are held in batch queues by `mbatchd`, which checks the load information on all candidate hosts periodically. When a host with the necessary resources becomes available, `mbatchd` sends the job to the `sbatchd` on that host for execution. When more than one host is

available, the best host is chosen. The `sbatchd` controls the execution of the jobs and reports job status to the `mbatchd`.

The LSF Batch Library (LSBLIB) is the Application Programming Interface (API) for LSF Batch, providing easy access to the services of `mbatchd` and `sbatchd`. LSBLIB provides a powerful interface for advanced users to develop new batch processing applications in C.

NQS Interoperation

NQS interoperation allows LSF users to submit jobs to remote NQS servers using the LSF user interface. The LSF administrator can configure LSF Batch queues to forward jobs to NQS queues. Users may then use any supported interface, including LSF Batch commands, `lsnqs` commands, and `xlsbatch`, to submit, monitor, signal and delete batch jobs in NQS queues. This feature provides users with a consistent user interface for jobs running under LSF Batch and NQS.

2. Getting Started

This chapter describes how to use some of the basic features of LSF. After following the examples in this chapter you should be able to use LSF for most of the everyday tasks.

Configuration options shown in the following examples, such as host types and model names, host CPU factors (representing relative processor speed), and resource names are examples only; your system likely has different values for these settings.

Getting Cluster Information

Cluster information includes the cluster master host, cluster name, cluster resource definitions, cluster administrator, etc.

Displaying the Cluster and Master Names

LSF provides tools for users to get information about the system. The first command you want to use when you learn LSF is `lsid`. This command tells you the version of LSF, the name of your LSF cluster, and the current master host.

```
% lsid
LSF 3.1, Dec 1, 1997
Copyright 1992-1997 Platform Computing Corporation

My cluster name is test_cluster
My master name is hostA
```

2 Getting Started

To find out who your cluster administrator is and see a summary of your cluster, run the `lsclusters` command:

```
% lsclusters
CLUSTER_NAME STATUS   MASTER_HOST   ADMIN   HOSTS   SERVERS
test_cluster ok       hostb        lsf     6       6
```

If you are using the LSF MultiCluster product, you will see one line for each of the clusters that your local cluster is connected to in the output of `lsclusters`.

Displaying Available Resources

The `lsinfo` command lists all the resources available in the cluster.

```
% lsinfo
RESOURCE_NAME  TYPE      ORDER  DESCRIPTION
r15s           Numeric   Inc    15-second CPU run queue length
rlm            Numeric   Inc    1-minute CPU run queue length (alias:cpu)
r15m           Numeric   Inc    15-minute CPU run queue length
ut             Numeric   Inc    1-minute CPU utilization (0.0 to 1.0)
pg            Numeric   Inc    Paging rate (pages/second)
io            Numeric   Inc    Disk IO rate (Kbytes/second)
ls            Numeric   Inc    Number of login sessions (alias: login)
it            Numeric   Dec    Idle time (minutes) (alias: idle)
tmp           Numeric   Dec    Disk space in /tmp (Mbytes)
swp           Numeric   Dec    Available swap space (Mbytes) (alias:swap)
mem           Numeric   Dec    Available memory (Mbytes)
ncpus         Numeric   Dec    Number of CPUs
ndisks        Numeric   Dec    Number of local disks
maxmem        Numeric   Dec    Maximum memory (Mbytes)
maxswp        Numeric   Dec    Maximum swap space (Mbytes)
maxtmp        Numeric   Dec    Maximum /tmp space (Mbytes)
cpuf          Numeric   Dec    CPU factor
rexpri        Numeric   N/A    Remote execution priority
server        Boolean   N/A    LSF server host
irix          Boolean   N/A    IRIX UNIX
hpux          Boolean   N/A    HP_UX
solaris       Boolean   N/A    Sun Solaris
cserver       Boolean   N/A    Compute server
fserver       Boolean   N/A    File server
```

aix	Boolean	N/A	AIX UNIX
type	String	N/A	Host type
model	String	N/A	Host model
status	String	N/A	Host status
hname	String	N/A	Host name

TYPE_NAME

HPPA
SGI6
ALPHA
SUNSOL
RS6K
NTX86

MODEL_NAME	CPU_FACTOR
DEC3000	10.00
R10K	14.00
PENT200	6.00
IBM350	7.00
SunSparc	6.00
HP735	9.00
HP715	5.00

The `lsinfo` command displays three lists of information:

- available resource names in the system
- available host types
- available host models

The resources listed by `lsinfo` include built-in resources maintained by the LIM and site specific resources configured by the LSF administrator. For a complete description of how LSF manages resources, see *'Resources' on page 35*.

The host types and host models are defined by the LSF administrator. Host types represent binary compatible hosts; all hosts of the same type can run the same executables. Host models give the relative CPU performance of different processors. In this example, your LSF cluster treats an R10K processor as being twice as fast as an IBM

2 Getting Started

350 processor (these numbers were invented for this example, and do not necessarily correspond to the actual performance of these systems).

Getting Host Information

LSF keeps information about all hosts in the cluster. Some information is static and some is dynamic. Static information is either configured by the LSF administrator, or is a fixed property of the system. An example of static host information is the amount of RAM memory available to users on a host.

Dynamic host information, or load indices, is determined by the LSF system, and updated regularly. Dynamic information represents the changing resources available on the host. Examples of dynamic host information are the current CPU load and the currently available temporary file space.

Displaying Static Host Information

A load sharing cluster may consist of hosts of differing architecture and speed. The `lshosts` command displays configuration information about hosts. All these parameters are defined by the LSF administrator in the LSF configuration files, or determined by the LIM directly from the system.

% `lshosts`

HOST_NAME	type	model	cpuf	ncpus	maxmem	maxswp	server	RESOURCES
hostD	SUNSOL	SunSparc	6.0	1	64M	112M	Yes	(solaris cserver)
hostB	ALPHA	DEC3000	10.0	1	94M	168M	Yes	(alpha cserver)
hostM	RS6K	IBM350	7.0	1	64M	124M	Yes	(cserver aix)
hostC	SGI6	R10K	14.0	16	1024M	1896M	Yes	(irix cserver)
hostA	HPPA	HP715	6.0	1	98M	200M	Yes	(hpux fserver)

In this example, the host type `SUNSOL` represents Sun SPARC systems running Solaris, and `ALPHA` represents a Digital Alpha server running Digital Unix.

See *'Listing Hosts'* on page 27 for a complete description of the `lshosts` command.

Displaying Load Information

The `lsload` command prints out current load information.

% `lsload`

HOST_NAME	status	r15s	r1m	r15m	ut	pg	ls	it	tmp	swp	mem
hostD	ok	0.1	0.0	0.1	2%	0.0	5	3	81M	82M	45M
hostC	ok	0.7	1.2	0.5	50%	1.1	11	0	322M	337M	252M
hostM	ok	0.8	2.2	1.4	60%	15.4	0	136	62M	57M	45M
hostA	busy	*5.2	3.6	2.6	99%	*34.44		0	70M	34M	18M
hostB	lockU	1.0	1.0	1.5	99%	0.8	5	33	12M	24M	23M

The first line lists the load index names, and each following line gives the load levels for one host. The `r15s`, `r1m` and `r15m` fields give the CPU load, averaged over different time intervals. The `ut` field gives the percentage of time the CPU is in use. `pg` is the paging rate, `ls` is the number of login sessions, `it` is the idle time (the time since the last interactive user activity), `swp` is the available swap space in megabytes, `mem` is the available RAM in megabytes, and `tmp` is the available temporary disk space in megabytes.

The `status` column gives the load status of the host. A host is busy if any load index is beyond its configured threshold. When a load index is beyond its threshold, it is printed with an asterisk '*'. In the above example, `hostA` is busy because load indices `r15s` and `pg` are too high. The `lshosts -l` command shows the load thresholds.

Hosts with `ok` status are listed first. The `ok` hosts are sorted based on CPU and memory load, with the best host listed first.

The `lsload` command reports more load indices if the `-l` option is given.

The `lsmon` command provides an updating display of load information. The `xlsmon` command is an X-windows graphical display of host status and load levels in your LSF cluster.

See the `lsload(1)`, `lsmon(1)`, and `xlsmon(1)` manual pages for more information. Also see '*Displaying the Load*' on page 29.

Running Jobs

LSF supports transparent execution of jobs on all server hosts in the cluster. You can run your program on the best available host and interact with it just as if it were running directly on your workstation. Keyboard signals such as `CTRL-Z` and `CTRL-C` work as expected.

Running Jobs on Remote Hosts

There are different ways to run jobs on a remote host. To run `myjob` on the best available host, enter:

```
% lsrun myjob
```

LSF automatically selects the best host that is of the same type as the local host.

If you want to run `myjob` on a host with specific resources, you must specify the resource requirements. For example,

```
% lsrun -R 'cserver && swp>100' myjob
```

runs `myjob` on a host that has resource `'cserver'` (see *'Displaying Available Resources' on page 14*) and has at least 100 megabytes of virtual memory available.

If you want to run your job on a particular host, use the `-m` option:

```
% lsrun -m hostD myjob
```

When you run an interactive job on a remote host, you can do most of the job controls as if it were running locally. If your shell supports job control, you can suspend and resume the job and bring the job to background or foreground as if it were a local job. For a complete description, see the `lsrun(1)` manual page.

You can also write one-line shell scripts or `csh` aliases to hide the remote execution. For example:

```
#!/bin/sh
# Script to remotely execute myjob
exec lsrun -m hostD myjob
```

or

```
% alias myjob "lsrun -m hostD myjob"
```

Load Sharing Commands With `lstdsh`

The `lstdsh` shell is a load-sharing version of the `tcsh` command interpreter. It is compatible with `csch` and supports many useful extensions. `csch` and `tcsh` users can use `lstdsh` to send jobs to other hosts in the cluster without needing to learn any new commands. You can run `lstdsh` from the command line, or use the `chsh` command to set it as your login shell. Refer to *'Using lstdsh' on page 149* for a more detailed description of `lstdsh`.

Parallel Processing With LSF Make

LSF Make is a load-sharing, parallel version of GNU make. It is compatible with makefiles for most versions of make. LSF Make uses the LSF load information to choose the best group of hosts for your make job. Targets in the makefile are processed in parallel on the chosen hosts using the LSF remote execution facilities. You do not need to modify your makefile to use LSF Make. By default, LSF Make chooses hosts that are all of the same type. LSF Make is invoked using the `lsmake` command.

The following example uses the `lsmake -V` and `-j 3` options to run on three hosts and produce verbose output:

```
% lsmake -V -j 3
[hostA] [hostD] [hostK]
<< Execute on local host >>
cc -O -c arg.c -o arg.o
<< Execute on remote host hostA >>
cc -O -c dev.c -o dev.o
<< Execute on remote host hostK >>
cc -O -c main.c -o main.o
<< Execute on remote host hostD >>
cc -O arg.o dev.o main.o
```

LSF Make includes control over parallelism for recursive makes, which are often used for source code trees that are organized into subdirectories. Parallelism can also be

2 Getting Started

controlled by the load on the NFS file server, so that parallel makes do not overload the server and slow everyone else down. See *‘Using LSF Make’ on page 159* for details.

Listing Hosts

LSF Batch uses some (or all) of the hosts in an LSF cluster as batch server hosts. The host list is configured by the LSF administrator. The `bhosts` command displays information about these hosts.

```
% bhosts
```

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
hostA	ok	-	2	1	1	0	0	0
hostB	ok	-	3	2	1	0	0	1
hostC	ok	-	32	10	9	0	1	0
hostD	ok	-	32	10	9	0	1	0
hostM	unavail	-	3	3	1	1	1	0

STATUS gives the status of `sbatchd`. If a host is down or its `sbatchd` is not up, its STATUS is ‘unavail’. The JL/U column shows the maximum number of job slots a single user can use on each host at one time. MAX gives the maximum number of job slots that are configured for each host. The RUN, SSUSP, and USUSP columns display the number of job slots in use by jobs in RUN state, suspended by the system, and suspended by the user, respectively. The field RSV shows job slots that are reserved by LSF Batch for some jobs. The NJOBS field shows the sum of field RUN, SSUSP, USUSP, and RSV.

For a more detailed description of the `bhosts` command see *‘Batch Hosts’ on page 79*.

Submitting a Job

To submit a job to the LSF Batch system, use the `bsub` command.

For example, submit the job `sleep 30`. This command does nothing, and takes 30 seconds to do it. The LSF administrator configures one queue to be the *default job queue*; if you submit a job without specifying a queue, the job goes to the default queue.

```
% bsub sleep 30
```

```
Job <1234> is submitted to default queue <normal>
```

In the above example, 1234 is the job ID assigned by LSF Batch to this job, and `normal` is the name of the default job queue.

Your batch job remains pending until all conditions for its execution are met. Each batch queue has execution conditions that apply to all jobs in the queue, and you can specify additional conditions when you submit the job.

The `-m "hostA hostB ..."` option specifies that the job must run on one of the specified hosts. By specifying a single host, you can force your job to wait until that host is available and then run on that host.

For a detailed description of the `bsub` command see ‘*Submitting Batch Jobs*’ on page 89.

Selecting a Job Queue

Job queues represent different job scheduling and control policies. All jobs submitted to the same queue share the same scheduling and control policy. Each job queue can use a configured subset of the server hosts in the LSF cluster; the default is to use all server hosts.

System administrators can configure job queues to control resource access by different users and types of application. Users select the job queue that best fits each job. The `bqueues` command lists the available LSF Batch queues:

% **bqueues**

QUEUE_NAME	PRIO	NICE	STATUS	MAX	JL/U	JL/P	NJOBS	PEND	RUN	SUSP
owners	49	10	Open:Active	-	-	-	1	0	1	0
priority	43	10	Open:Active	10	-	-	8	5	3	0
night	40	10	Open:Inactive	-	-	-	44	44	0	0
short	35	20	Open:Active	20	-	2	4	0	4	0
license	33	10	Open:Active	40	-	-	1	1	0	0
normal	30	20	Open:Active	-	2	-	0	0	0	0

A dash ‘-’ in any entry means that the column does not apply to the row. In this example some queues have no per-queue, per-user or per-processor job limits configured, so the `MAX`, `JL/U` and `JL/P` entries are ‘-’.

You can submit jobs to a queue as long as its `STATUS` is `Open`. However, jobs are not dispatched unless the queue is `Active`.

2 Getting Started

Tracking Batch Jobs

The `bjobs` command reports the status of LSF Batch jobs. The `-u all` option specifies that jobs for all users should be listed; the default is to list only jobs you submitted. Running jobs are listed first. Pending jobs are listed in the order in which they will be scheduled. Jobs in high priority queues are listed before those in lower priority queues.

```
% bjobs -u all
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
1004	user	RUN	short	hostA	hostA	job0	Dec 16 09:23
1235	user2	PEND	priority	hostM		job1	Dec 11 13:55
1234	user2	SSUSP	normal	hostD	hostM	job3	Dec 11 10:09
1250	user1	PEND	short	hostA		job4	Dec 11 13:59

If you also want to see jobs that finished recently, enter:

```
% bjobs -a
```

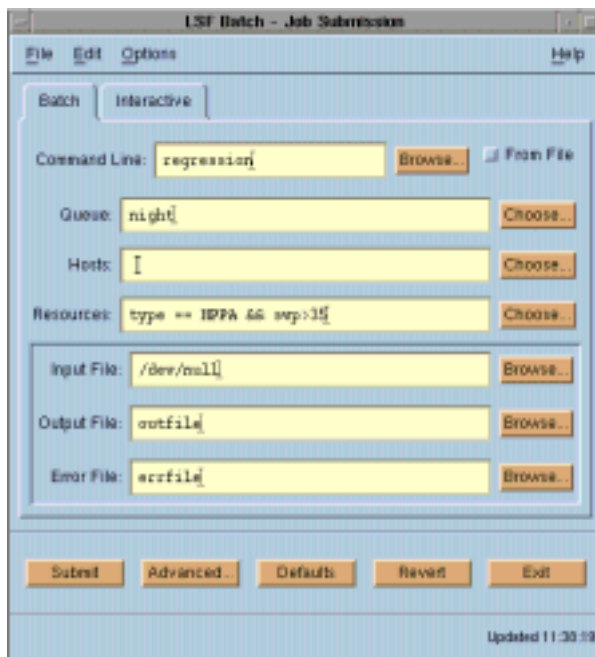
All your jobs that are still in the LSF Batch system and jobs finished recently are displayed.

The `bjobs` command has many other options. See '*Batch Jobs*' on page 56. Also refer to the `bjobs(1)` manual page for a complete description.

xbsub and xlsbatch GUI Applications

You can submit your job to the LSF Batch system using the graphical user interface application `xbsub` as shown in *Figure 3*.

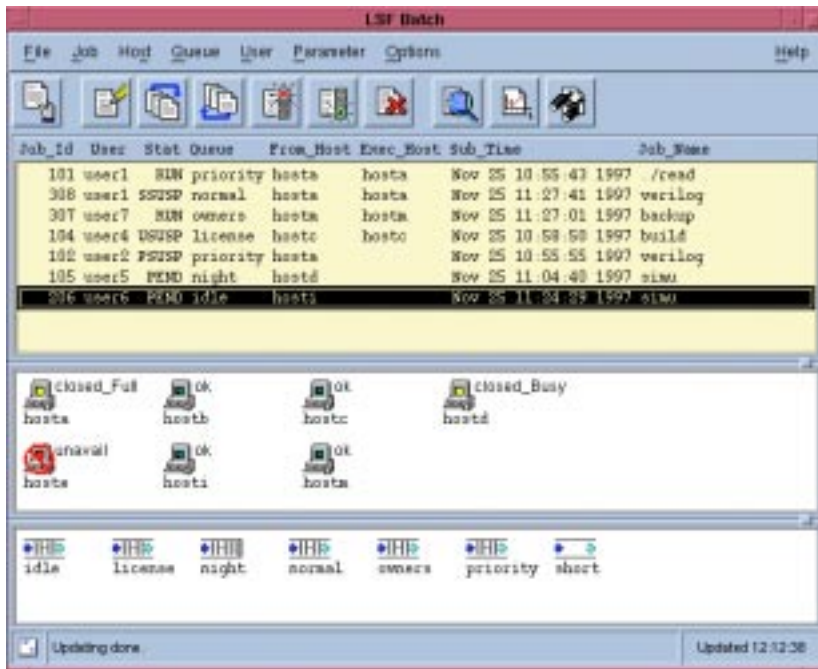
Figure 3. `xbsub` Job Submission Window



2 Getting Started

xlsbatch is another graphical user interface application for LSF Batch. You can use it to monitor host, job, and queue status, and control your jobs.

Figure 4. xlsbatch Main Window



Both xbsub and xlsbatch have extensive on-line help available through the Help menu of each application.

xbsub can be started either directly from the command line or from xlsbatch using the 'Submit' button.

3. Cluster Information

This chapter is a detailed tutorial on how to use the LSF commands that report information about the LSF cluster. Cluster information includes:

- cluster status
- resource configuration
- host configuration
- host load levels

Cluster information is available through commands and a graphical user interface.

Finding the Master

The `lsid` command tells you the version of LSF, the name of the load-sharing cluster, and the current master host.

```
% lsid
LSF 3.1, Dec 1, 1997
Copyright 1992-1997 Platform Computing Corporation

My cluster name is sample_cluster
My master name is hostA
```

Listing Resources

The `lsinfo` command displays the resources, host types, and host models in the LSF cluster. See ‘*Displaying Available Resources*’ on page 14 for an example of the `lsinfo` command.

The `-l` option to the `lsinfo` command displays all information available about load indices. You can also specify load indices on the command line to display information about the selected indices:

```
% lsinfo -l swp cs irix
RESOURCE_NAME:  swp
DESCRIPTION: Available swap space (Mbytes) (alias: swap)
TYPE           ORDER   INTERVAL  BUILTIN  DYNAMIC
Numeric        Dec     60        Yes       Yes

RESOURCE_NAME:  cpuf
DESCRIPTION: CPU factor
TYPE           ORDER   INTERVAL  BUILTIN  DYNAMIC
Numeric        Dec     0         Yes       No

RESOURCE_NAME:  irix
DESCRIPTION: IRIX UNIX
TYPE           ORDER   INTERVAL  BUILTIN  DYNAMIC
Boolean        N/A     0         No        No
```

TYPE

Indicates whether the resource is numeric, string, or Boolean.

ORDER

`Inc` if the numeric value of the load index increases as the load it measures increases, such as `ut` (CPU utilization). `Dec` if the numeric value decreases as the load increases. `N/A` if the resource is not numeric.

INTERVAL

The load index is updated every `INTERVAL` seconds. a value of 0 means the value never changes.

BUILTIN

If **BUILTIN** is **Yes**, the resource name is defined internally by the LIM. If **BUILTIN** is **No**, the resource name is site-specific defined by the LSF administrator.

DYNAMIC

If **DYNAMIC** is **Yes** the resource value changes over time. If **DYNAMIC** is **No** the resource represents information that does not change such as the total swap space in the host.

Listing Hosts

A load-sharing cluster may consist of hosts of different architecture and speeds. The `lshosts` command displays static information about hosts.

% lshosts

HOST_NAME	type	model	cpuf	ncpus	maxmem	maxswp	server	RESOURCES
hostD	SUNSOL	SunSparc	6.0	1	64M	112M	Yes	(solaris cserver)
hostB	ALPHA	DEC3000	10.0	1	94M	128M	Yes	(alpha cserver)
hostM	RS6K	IBM350	7.0	1	64M	124M	Yes	(cserver aix)
hostC	SGI6	ORIGIN2K	14.0	16	1024M	1896M	Yes	(irix cserver)
hostA	HPPA	HP715	6.0	1	98M	200M	Yes	(hpux fserver)

The fields displayed are:

type - the host CPU architecture. Hosts that can run the same binary programs should have the same **type**.

model and **cpuf** - the host model and relative CPU performance factor. The higher the number, the faster the CPU.

ncpus - the number of processors on this host.

maxmem - the maximum amount of physical memory available for user processes.

maxswp - the total swap space available.

3 Cluster Information

server - Yes if the host is an LSF server, No if the host is a client.

RESOURCES - the boolean resources defined for this host.

All these parameters are defined by the LSF administrator in the LSF configuration files, or determined directly from the system.

The **-l** option to the **lshosts** command displays more detailed information about hosts, including the load thresholds. You can also use the **-R *resreq*** option to display hosts with specific resources, or specify hosts on the command line.

There is also the **-w** option which displays information in table format without truncating the fields. This is especially useful for scripts that parse the output of **lshosts**.

```
% lshosts -l hostC
```

```
HOST_NAME: hostC
```

type	model	cpuf	ncpus	ndisks	maxmem	maxswp	maxtmp	rexpri	server
SGI6	ORIGIN2K	14	16	4	1024M	1896M	120M	0	Yes

```
RESOURCES: irix cserver
```

```
RUN_WINDOWS: (always open)
```

```
LOAD_THRESHOLDS:
```

r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
-	3.5	-	-	-	-	-	-	-	974M	451M

The extra fields displayed by the **-l** option are:

ndisks - the number of disk drives directly attached to the host

maxtmp - the size of the disk partition that contains the **/tmp** directory

UNIX

rexpri - the execution priority of remote jobs run by the RES. **rexpri** is a number between -20 and 20, with -20 representing the highest priority and 20 the lowest. The default **rexpri** is 0, which corresponds to the default scheduling priority of 0 on BSD-based UNIX systems and 20 on System V-based systems.

RUN_WINDOWS

The time windows during which LIM considers the host as available to execute remote jobs.

Note

These run windows have the same function for LSF hosts as dispatch windows have for LSF Batch hosts.

LOAD_THRESHOLDS

The host is considered busy if any load index is above its threshold (or below, for decreasing load indices). LSF does not send interactive jobs to busy hosts. If the threshold is displayed as a dash '-', the value of that load index does not affect the host's status.

Displaying the Load

The `lsload` command prints out current load information.

% lsload

HOST_NAME	status	r15s	r1m	r15m	ut	pg	ls	it	tmp	swp	mem
hostD	ok	0.1	0.0	0.1	2%	0.0	5	3	81M	82M	45M
hostC	ok	0.7	1.2	0.5	50%	1.1	11	0	322M	337M	252M
hostM	ok	0.8	2.2	1.4	60%	15.4	0	136	62M	37M	44M
hostA	busy	*5.2	3.6	2.6	99%	*34.4	4	0	70M	34M	18M
hostB	lockU	1.0	1.0	1.5	99%	0.8	5	33	12M	24M	23M

The first line lists all the load index names. The load indices are described in '*Load Indices*' on page 37.

The `status` column gives the load status of the host. A host is busy if any load index is beyond its configured threshold. When a load index is beyond its threshold, it is printed with an asterisk '*'. In the above example, `hostA` is busy because load indices `r15s` and `pg` are too high.

By default hosts are sorted based on CPU and memory load, with the best host listed first. You can specify an order string using the `-R resreq` option to sort the hosts in other ways.

3 Cluster Information

The `-l` option displays the values of all load indices, including external load indices. You can also specify host names on the command line to display the load of specific hosts. In this example `nio` is an external load index defined by the LSF administrator.

```
% lsload -l hostM
```

HOST_NAME	status	r15s	rlm	r15m	ut	pg	io	ls	it	tmp	swp	mem	nio
hostM	lockW	0.7	0.5	0.1	21%	0.0	228	7	0	31M	52M	25M	3.5

The `-l` option displays the full host name, rather than truncating the name to fit in a limited screen width. The output from `lsload -l` is better suited for automatic processing, since the host name is always complete. It also shows all the load indices rather than just some of them.

The `lsmon` command provides an updating display of load information. An example display from `lsmon` is shown below. You can specify the resource requirements, refresh interval, and other parameters interactively or on the command line. See the `lsmon(1)` manual page for more information.

```
% lsmon hostA hostB
```

```
Hostname: hostA
```

```
Refresh rate: 10 secs
```

HOST_NAME	status	r15s	rlm	r15m	ut	pg	ls	it	swp	mem	tmp
hostB	ok	0.0	0.0	0.0	6%	0.0	7	45	82M	51M	206M
hostA	ok	0.2	0.2	0.1	5%	8.9	2	0	140M	70M	119M

Graphical Load Display

`xlsmon` is a graphical user interface (GUI) application that displays host status, load levels, load history, and LSF cluster configuration information. The `xlsmon` main window shows an icon for each host in the cluster, with each host labelled with its

status. Hosts change colour as their status changes. The `xlsmon` main window is shown below.

Figure 5. `xlsmon` Main Window



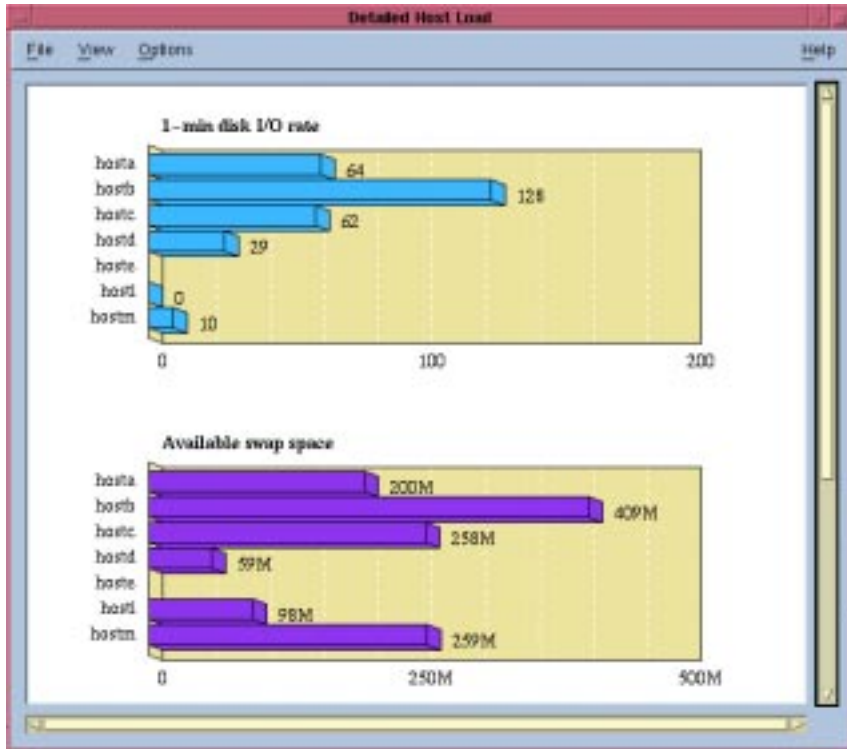
Each `xlsmon` window has a “Help” menu item that calls up online help. For more information about using `xlsmon`, see the online help.

You can choose other displays from the “View” menu.

3 Cluster Information

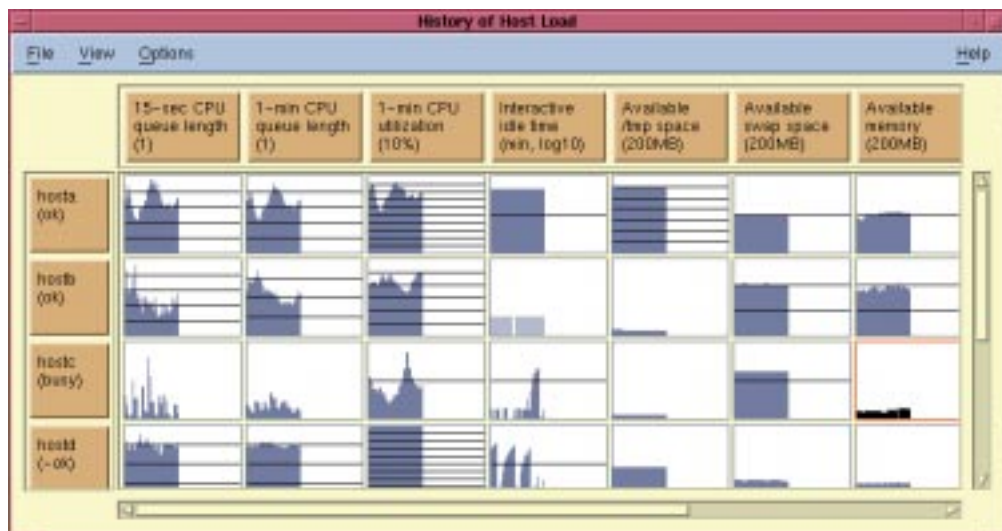
The 'Detailed Host Load' window displays load levels as bar graphs. You can select which load indices and which hosts are displayed by choosing options from the "View" menu in the 'Detailed Host Load' window.

Figure 6. xlsmon Detailed Host Load Window



The 'History of Host Load' window displays the load levels as stripcharts, so you can see the load history starting from when the 'History of Host Load' window is first displayed. As with the 'Detailed Host Load' window, you can select hosts and load indices by choosing options from the "View" menu.

Figure 7. xlsmon History of Host Load Window



The 'Cluster Configuration' window below, shows the same host information as the `lshosts` command displays.

Figure 8. xlsmon Cluster Configuration Window

Cluster Configuration								
HOST NAME	type	model	cpuf	ncpus	maxmem	maxswap	server	RESOURCES
hosta	HPPA	HP715	13.0	4	1204M	2999M	Yes	Not defined
hostb	ALPHA	DEC3000	20.0	4	402M	768M	Yes	alpha
hostc	SGIS	ORIGIN2K	20.0	1	88M	384M	Yes	lgux
hostd	SUNSC1	SunSparc	12.0	1	58M	224M	Yes	Not defined
hoste	HPPA30	HP715	14.0	-	-	-	Yes	lgux
hostf	HPPA	HP715	13.0	1	63M	125M	Yes	lgux
hostg	386K	IBM590	18.0	1	82M	390M	Yes	Not defined

4. Resources

This chapter describes the system resources LSF keeps track of and how you use LSF resource specifications. Topics covered in this chapter are:

- resources and load indices
- specifying resource requirements
- using task lists to set resource requirements for applications

Introduction to Resources

A computer may be thought of as a collection of resources used to execute programs. Different applications often require different resources. For example, a number crunching program may take a lot of CPU power, but a large spreadsheet may need a lot of memory to run well. Some applications may run only on machines of a specific type, and not on others. To run applications as efficiently as possible, the LSF system needs to take these factors into account.

In LSF, resources are handled by naming them and tracking information relevant to them. LSF does its scheduling according to application's resource requirements and resources available on individual hosts. LSF classifies resources in different ways.

Classification by Availability

general resources

These are resources that are available on all hosts, e.g. all the load indices, number of processors on a host, total swap space, host status.

4 Resources

special resources

These are resources that are only associated with some hosts, e.g. FileServer, aix, solaris, SYSV.

Classification by the Way Values Change

dynamic resources

These are resources that change their values dynamically, e.g. all the load indices, host status.

static resources

These are resources that do not change their values, e.g. all resources except load indices and host status are static resources.

Classification by Types of Values

numerical resources

These are resources that take numerical values, e.g. all the load indices, number of processors on a host, host CPU factor.

string resources

These are resources that take string values, e.g. host type, host model, host status.

Boolean resources

These are resources that denote the availability of specific features, e.g. hspice, FileServer, SYSV, aix.

Classification by Definition

configured resources

These are resources defined by user sites, such as external indices and resources defined in the `lsf.shared` file, e.g. FileServer, fddi.

built-in resources

These are resources that are always defined by LIM, e.g. load indices, number of CPUs, total swap space.

Classification by Location

host-based resources

These are resources that are not shared among hosts, but are tied to individual hosts. An application must run on a particular host to access such resources, e.g. CPU, memory (using up memory on one host does not affect the available memory on another host), swap space.

shared resources

These are resources that are not associated with individual hosts in the same way, but are "owned" by the entire cluster, or a subset of hosts within the cluster. An application can access such a resource from any host which is configured to share it, but doing so affects its value as seen by other hosts, e.g. floating licenses, shared file systems.

Resource names are case sensitive, and can be up to 29 characters in length (excluding some characters reserved as operators in resource requirement strings). You can list the resources available in your cluster using the `lsinfo` command.

Load Indices

Load indices measure the availability of dynamic, non-shared resources on hosts in the LSF cluster. Load indices built into the LIM are updated at fixed time intervals. *External load indices* are updated when new values are received from the external load collection program, ELIM, configured by the LSF administrator. Load indices are numeric in value.

4 Resources

Table 1 summarizes the load indices collected by the LIM.

Table 1. Load Indices

Index	Measures	Units	Direction	Averaged over	Update Interval
status	host status	string			15 seconds
r15s	run queue length	processes	increasing	15 seconds	15 seconds
r1m	run queue length	processes	increasing	1 minute	15 seconds
r15m	run queue length	processes	increasing	15 minutes	15 seconds
ut	CPU utilisation	(per cent)	increasing	1 minute	15 seconds
pg	paging activity	pages in + pages out per second	increasing	1 minute	15 seconds
ls	logins	users	increasing	N/A	30 seconds
it	idle time	minutes	decreasing	N/A	30 seconds
swp	available swap space	megabytes	decreasing	N/A	15 seconds
mem	available memory	megabytes	decreasing	N/A	15 seconds
tmp	available space in temporary filesystem ^a	megabytes	decreasing	N/A	120 seconds
io	disk I/O (shown by <code>lsload -l</code>)	kilobytes per second	increasing	1 minute	15 seconds
name	external load index configured by LSF administrator				site defined

a. Directory C:\temp on NT and /tmp on UNIX.

The `status` index is a string indicating the current status of the host. This status applies to the LIM and RES. The possible values for `status` are:

ok

The LIM can select the host for remote execution

busy

A load index exceeds the threshold defined by the LSF administrator; the LIM will not select the host for interactive jobs

lockU

The host is locked by a user or the LSF administrator

lockW

The host's availability time window is closed

unavail

The LIM on the host is not responding

unlicensed

The host does not have a valid LSF license.

If the LIM is available but the RES server is not responding, `status` begins with a '-'.

Here is an example of the output from `lsload`:

% **lsload**

HOST_NAME	status	r15s	r1m	r15m	ut	pg	ls	it	tmp	swp	mem
hostN	ok	0.0	0.0	0.1	1%	0.0	1	224	43M	67M	3M
hostK	-ok	0.0	0.0	0.0	3%	0.0	3	0	38M	40M	7M
hostG	busy	*6.2	6.9	9.5	85%	1.1	30	0	5M	400M	385M
hostF	busy	0.1	0.1	0.3	7%	*17	6	0	9M	23M	28M
hostV	unavail										

The `r15s`, `r1m` and `r15m` load indices are the 15-second, 1-minute and 15-minute average CPU run queue lengths. This is the average number of processes ready to use the CPU during the given interval.

UNIX

Note

Run queue length indices are not necessarily the same as the load averages printed by the `uptime(1)` command; uptime load averages on some platforms also include processes that are in short term wait states (such as paging or disk I/O).

4 Resources

On multiprocessor systems more than one process can execute at a time. LSF scales the run queue value on multiprocessor systems to make the CPU load of uniprocessors and multiprocessors comparable. The scaled value is called the *effective run queue length*. The `-E` option shows the effective run queue length.

LSF also adjusts the CPU run queue based on the relative speeds of the processors (the CPU factor). The *normalized run queue length* is adjusted for both number of processors and CPU speed. The host with the lowest normalized run queue length will run a CPU intensive job the fastest. The `-N` option shows the normalized CPU run queue lengths.

The `ut` index measures CPU utilization, which is the percentage of time spent running system and user code. A host with no process running has a `ut` value of 0 percent; a host on which the CPU is completely busy has a `ut` of 100 percent.

The `pg` index gives the virtual memory paging rate in pages per second. This index is closely tied to the amount of available RAM memory and the total size of the processes running on a host; if there is not enough RAM to satisfy all processes, the paging rate will be high. Paging rate is a good measure of how a machine will respond to interactive use; a machine that is paging heavily feels very slow.

The paging rate is reported in units of pages rather than kilobytes, because the relationship between interactive response and paging rate is largely independent of the page size.

The `ls` index gives the number of users logged in. Each user is counted once, no matter how many times they have logged into the host.

The `it` index is the interactive idle time of the host, in minutes. Idle time is measured from the last input or output on a directly attached terminal or a network pseudo-terminal supporting a login session.

UNIX Note

This does not include activity directly through the X server such as CAD applications or emacs windows, except on SunOS 4, Solaris, and HP-UX systems.

The `tmp` index is the space available on the file system that contains the `/tmp` (UNIX) or the `C:\temp` (NT) directory in megabytes.

The `swp` index gives the currently available swap space in megabytes. This represents the largest process that can be started on the host.

The `mem` index is an estimate of the real memory currently available to user processes. This represents the approximate size of the largest process that could be started on a host without causing the host to start paging. This is an approximation because the virtual memory behaviour of operating systems varies from system to system and is hard to predict.

The `io` index is only displayed with the `-l` option to `lsload`. This index measures I/O throughput to disks attached directly to this host, in kilobytes per second. It does not include I/O to disks that are mounted from other hosts.

External load indices are defined by the LSF administrator. The `lsinfo` command lists the external load indices and the `lsload -l` command displays the values of all load indices. If you need more information about the external load indices defined at your site, contact your LSF administrator.

Static Resources

Static resources represent host information that does not change over time such as the maximum RAM available to user processes and the number of processors in a machine. Most static resources are determined by the LIM at start-up time. *Table 2* lists the static resources reported by the LIM.

4 Resources

Table 2. Static Resources

Index	Measures	Units	Determined by
<code>type</code>	host type	string	configuration
<code>model</code>	host model	string	configuration
<code>hname</code>	host name	string	configuration
<code>cpuf</code>	CPU factor	relative	configuration
<code>server</code>	host can run remote jobs	Boolean	configuration
<code>rexpri</code>	execution priority (UNIX only)	<code>nice(2)</code> argument	configuration
<code>ncpus</code>	number of processors	processors	LIM
<code>ndisks</code>	number of local disks	disks	LIM
<code>maxmem</code>	maximum RAM memory available to users	megabytes	LIM
<code>maxswp</code>	maximum available swap space	megabytes	LIM
<code>maxtmp</code>	maximum available space in temporary file system	megabytes	LIM

The `type` and `model` static resources are strings specifying the host type and model.

The CPU factor is the speed of the host's CPU relative to other hosts in the cluster. If one processor is twice the speed of another, its CPU factor should be twice as large. The CPU factors are defined by the LSF administrator. For multiprocessor hosts the CPU factor is the speed of a single processor; LSF automatically scales the host CPU load to account for additional processors.

The `server` static resource is Boolean; its value is 1 if the host is configured to execute tasks from other hosts, and 0 if the host is a client.

Static resources can be used to select appropriate hosts for particular jobs based on binary architecture, relative CPU speed, and system configuration.

Shared Resources

A shared resource is a resource that is not tied to a specific host, but is associated with the entire cluster, or a specific subset of hosts within the cluster. Examples of shared resources include:

- floating licenses for software packages
- disk space on a file server which is mounted by several machines
- the physical network connecting the hosts

An application may use a shared resource by running on any host from which that resource is accessible. For example, in a cluster in which each host has a local disk but can also access a disk on a file server, the disk on the file server is a shared resource, and the local disk is a host-based resource. There will be one value for the entire cluster which measures the utilization of the shared resource, but each host-based resource is measured separately.

LSF does not contain any built-in shared resources. All shared resources must be configured by the LSF Administrator. A shared resource may be configured to be dynamic or static. In the above example, the total space on the shared disk may be static while the amount of space currently free is dynamic. A site may also configure the shared resource to report numeric, string or Boolean values.

Viewing Shared Resources

In order to view the shared resources in the cluster, use the `-s` option of the `lshosts`, `lsload`, and `bhosts` commands. For example, suppose a cluster consists of two hosts, each of which have access to a total of five floating licenses for a particular package. They also access a scratch directory, containing 500MB of disk space, from a file server. The LSF administrator has set the resource definitions as shown in *Table 3*.

4 Resources

Table 3. Example of Shared Resources

Resource Name	Describes
tot_lic	Total number of licenses in cluster
tot_scratch	Total amount of space in shared scratch directory (in MB)
avail_lic	Currently available number of licenses
avail_scratch	Currently available space in shared scratch dir (in MB)

The output of `lshosts -s` could be:

```
% lshosts -s
```

```
RESOURCE      VALUE  LOCATION
tot_lic        5      host1 host2
tot_scratch    500    host1 host2
```

The “VALUE” field indicates the amount of that resource. The “LOCATION” column shows the hosts which share this resource. The information displayed by `lshosts(1)` is static, meaning that the value will not change over time. `lsload -s` displays the information about shared resources which are dynamic:

```
% lsload -s
```

```
RESOURCE      VALUE      LOCATION
avail_lic      2          host1 host2
avail_scratch  100        host1 host2
```

The above output indicates that 2 licenses are available, and that the shared scratch directory currently contains 100MB of space.

Under LSF Batch, shared resources may be viewed using `bhosts -s`:

```
% bhosts -s
```

```
RESOURCE      TOTAL      RESERVED    LOCATION
tot_lic        5          0.0         hostA hostB
tot_scratch    500        0.0         hostA hostB
avail_lic      2          3.0         hostA hostB
avail_scratch  100        400.0       hostA hostB
```

The “TOTAL” column gives the value of the resource. For dynamic resources, the “RESERVED” column displays the amount that has been reserved by running jobs.

Boolean Resources

Boolean resource names are used to describe features that may be available only on some machines in a cluster. For example:

- Machines may have different types and versions of operating systems.
- Machines may play different roles in the system such as file server or compute server.
- Some machines may have special-purpose devices needed by some applications.
- Certain software packages or licenses may be available only on some of the machines.

Any characteristics or attributes of certain hosts that can be useful in selecting hosts for remote jobs may be configured as Boolean resources. Specifying a Boolean resource in the resource requirements of a job limits the set of computers that can execute the job. *Table 4* lists some examples of Boolean resources.

Table 4. Examples of Boolean Resources

Resource Name	Describes	Meaning of Example Name
cs	role in cluster	compute server
fs	role in cluster	file server
solaris	operating system	Solaris operating system
frame	available software	FrameMaker license

Listing Resources

The `lsinfo` command lists all the resources configured in the LSF cluster. See *'Displaying Available Resources' on page 14* for an example of the `lsinfo` command. The `lsinfo -l` option gives more detail about each index:

```
% lsinfo -l rlm
RESOURCE_NAME:  rlm
DESCRIPTION:  1-minute CPU run queue length (alias: cpu)
TYPE          ORDER      INTERVAL  BUILTIN   DYNAMIC
Numeric       Inc         15        Yes       Yes
```

- `TYPE` indicates whether the resource is numeric, string, or Boolean.
- `ORDER` is `Inc` if the numeric value of the load index increases as the load it measures increases, such as `ut` (CPU utilization), or `Dec` if the numeric value decreases as the load increases. If the resource is not numeric, the `ORDER` is `N/A`.
- `INTERVAL` shows the number of seconds between updates of that index.
- `BUILTIN` is `Yes` if the index is built into the LIM and `No` if the index is external.
- `DYNAMIC` is `Yes` if the resource is a load index that changes over time and `No` if the resource is a static or Boolean resource.

Resource Requirement Strings

A resource requirement string describes the resources a job needs. LSF uses resource requirements to select hosts for remote execution and job execution.

A resource requirement string is divided into four sections:

- A *selection* section.
- An *ordering* section.

- A *resource usage* section.
- A *job spanning* section

The selection section specifies the criteria for selecting hosts from the system. The ordering section indicates how the hosts that meet the selection criteria should be sorted. The resource usage section specifies the expected resource consumption of the task. The job spanning section indicates if a (parallel) batch job should span across multiple hosts.

The syntax of a resource requirement expression is:

```
select[selectstring] order[orderstring] rusage[usagestring] span
[spanstring]
```

The section names are *select*, *order*, *rusage*, and *span*. The syntax for each of *selectstring*, *orderstring*, *usagestring*, and *spanstring* is defined below.

Note

The square brackets are an essential part of the resource requirement expression.

Depending on the command, one or more of these sections may apply. The `lshosts` command only selects hosts, but does not order them. The `lsload` command selects and orders hosts, while `lsplace` uses the information in *select*, *order*, and *rusage* sections to select an appropriate host for a task. The `lsloadadj` command uses the *rusage* section to determine how the load information should be adjusted on a host, while `bsub` uses all four sections. Sections that do not apply for a command are ignored.

If no section name is given, then the entire string is treated as a selection string. The *select* keyword may be omitted if the selection string is the first string in the resource requirement.

Selection String

The selection string specifies the characteristics a host must have to match the resource requirement. It is a logical expression built from a set of resource names. The `lsinfo` command lists all the resource names and their descriptions. The resource names `swap`, `idle`, `logins`, and `cpu` are accepted as aliases for `swp`, `it`, `ls`, and `r1m` respectively.

4 Resources

The selection string can combine resource names with logical and arithmetic operators. Non-zero arithmetic values are treated as logical TRUE, and zero as logical FALSE. Boolean resources (for example, `server` to denote LSF server hosts) have a value of one if they are defined for a host, and zero otherwise.

Table 5 shows the operators that can be used in selection strings. The operators are listed in order of decreasing precedence.

Table 5. Operators in Resource Requirements

Syntax	Meaning
<code>-a</code> <code>!a</code>	Negative of <code>a</code> Logical not: 1 if <code>a==0</code> , 0 otherwise
<code>a * b</code> <code>a / b</code>	Multiply <code>a</code> and <code>b</code> Divide <code>a</code> by <code>b</code>
<code>a + b</code> <code>a - b</code>	Add <code>a</code> and <code>b</code> Subtract <code>b</code> from <code>a</code>
<code>a > b</code> <code>a < b</code> <code>a >= b</code> <code>a <= b</code>	1 if <code>a</code> is greater than <code>b</code> , 0 otherwise 1 if <code>a</code> is less than <code>b</code> , 0 otherwise 1 if <code>a</code> is greater than or equal to <code>b</code> , 0 otherwise 1 if <code>a</code> is less than or equal to <code>b</code> , 0 otherwise
<code>a == b</code> <code>a != b</code>	1 if <code>a</code> is equal to <code>b</code> , 0 otherwise 1 if <code>a</code> is not equal to <code>b</code> , 0 otherwise
<code>a && b</code>	Logical AND: 1 if both <code>a</code> and <code>b</code> are non-zero, 0 otherwise
<code>a b</code>	Logical OR: 1 if either <code>a</code> or <code>b</code> is non-zero, 0 otherwise

The selection string is evaluated for each host; if the result is non-zero, then that host is selected. For example:

```
select[(swp > 50 && type == MIPS) || (swp > 35 && type == ALPHA)]
```

```
select[((2*r15s + 3*r1m + r15m) / 6 < 1.0) && !fs && (cpuf > 4.0)]
```

For the string resources `type` and `model`, the special value `any` selects any value and `local` selects the same value as that of the local host. For example, `type==local` selects hosts of the same type as the host submitting the job. If a job can run on any type

of host, include `type==any` in the resource requirements. If no `type` is specified, the default depends on the command. For `lshosts`, `lsload`, `lsmon` and `lslogin` the default is `type==any`. For `lsplace`, `lsrun`, `lsgrun`, and `bsub` the default is `type==local` unless a model or Boolean resource is specified, in which case it is `type==any`.

Order String

The order string allows the selected hosts to be sorted according to the values of resources. The syntax of the order string is

```
[ - ]res[ : [ - ]res ] . . .
```

Each *res* must be a dynamic load index; that is, one of the indices `r15s`, `r1m`, `r15m`, `ut`, `pg`, `io`, `ls`, `it`, `tmp`, `swp`, `mem`, or an external load index defined by the LSF administrator. For example, `swp:r1m:tmp:r15s` is a valid order string.

Note

The values of `r15s`, `r1m`, and `r15m` used for sorting are the normalized load indices returned by `lsload -N` (see ‘Load Indices’ on page 37).

The order string is used for host sorting and selection. The ordering begins with the rightmost index in the order string and proceeds from right to left. The hosts are sorted into order based on each load index, and if more hosts are available than were requested, the LIM drops the least desirable hosts according to that index. The remaining hosts are then sorted by the next index.

After the hosts are sorted by the leftmost index in the order string, the final phase of sorting orders the hosts according to their status, with hosts that are currently not available for load sharing (that is, not in the `ok` state) listed at the end.

Because the hosts are resorted for each load index, only the host status and the leftmost index in the order string actually affect the order in which hosts are listed. The other indices are only used to drop undesirable hosts from the list.

When sorting is done on each index, the direction in which the hosts are sorted (increasing vs decreasing values) is determined by the default order returned by `lsinfo` for that index. This direction is chosen such that after sorting, the hosts are ordered from best to worst on that index.

4 Resources

When an index name is preceded by a minus sign '-', the sorting order is reversed so that hosts are ordered from worst to best on that index.

The default sorting order is `r1m:pg` (except for `lslogin(1):ls:r1m`).

Resource Usage String

This string defines the expected resource usage of the task. It is used to specify resource reservations for LSF Batch jobs, or for mapping tasks onto hosts and adjusting the load when running interactive jobs.

LSF Batch Jobs

For LSF Batch jobs, the resource usage section is used along with the queue configuration parameter `RES_REQ` (see '*Scheduling Conditions*' on page 65). External indices are also considered in the resource usage string.

The syntax of the resource usage string is

```
res=value[:res=value]...[:res=value][:duration=value][:decay=valu  
e]
```

The *res* parameter can be any load index. The *value* parameter is the initial reserved amount. If *res* or *value* is not given, the default is not to reserve that resource.

The *duration* parameter is the time period within which the specified resources should be reserved. It is specified in minutes by default. If the value is followed by the letter 'h', it is specified in hours. For example, '*duration*=30' and '*duration*=2h' specify a duration of 30 minutes and two hours respectively. If *duration* is not specified, the default is to reserve the total amount for the lifetime of the job.

The *decay* parameter indicates how the reserved amount should decrease over the *duration*. A value of 1, '*decay*=1', indicates that system should linearly decrease the amount reserved over the duration. The default *decay* value is 0, which causes the total amount to be reserved for the entire *duration*. Values other than 0 or 1 are unsupported. If *duration* is not specified *decay* is ignored.

```
rusage[mem=50:duration=100:decay=1]
```

The above example indicates that 50MB memory should be reserved for the job. As the job runs, the amount reserved will decrease at approximately 0.5 megabytes per minute until the 100 minutes is up.

LSF Base Jobs

Resource reservation is only available for LSF Batch. If you run jobs using LSF Base, such as through `lsrun`, LIM uses resource usage to determine the placement of jobs. LIM's placement is limited in comparison to LSF Batch in that the LIM does not track when an application finishes. Resource usage requests are used to temporarily increase the load so that a host is not overloaded. When LIM makes a placement advice, external load indices are not considered in the resource usage string. In this case, the syntax of the resource usage string is

```
res[=value]:res[=value]: ... :res[=value]
```

The *res* is one of the resources whose value is returned by the `lsload` command.

```
rusage[r1m=0.5:mem=20:swp=40]
```

The above example indicates that the task is expected to increase the 1-minute run queue length by 0.5, consume 20 Mbytes of memory and 40 Mbytes of swap space.

If no value is specified, the task is assumed to be intensive in using that resource. In this case no more than one task will be assigned to a host regardless of how many CPUs it has.

The default resource usage for a task is `r15s=1.0:r1m=1.0:r15m=1.0`. This indicates a CPU intensive task which consumes few other resources.

Job Spanning String

This string specifies the locality of a parallel job (see '*Specifying Locality*' on page 104). Currently only the following two cases are supported:

```
span[hosts=1]
```

This indicates that all the processors allocated to this job must be on the same host.

```
span[ptile=n]
```

4 Resources

This indicates that only n processors on each host should be allocated to the job regardless of how many processors the host possesses.

If span is omitted, LSF Batch will allocate the required processors for the job from the available set of processors.

Specifying Shared Resources

A shared resource may be used in the resource requirement string of any LSF command. For example when submitting an LSF Batch job which requires a certain amount of shared scratch space, you might submit the job as follows:

```
% bsub -R "avail_scratch > 200 && swap > 50" myjob
```

The above assumes that all hosts in the cluster have access to the shared scratch space. The job will only be scheduled if the value of the "avail_scratch" resource is more than 200MB and will go to a host with at least 50MB of available swap space.

It is possible for a system to be configured so that only some hosts within the LSF cluster have access to the scratch space. In order to exclude hosts which cannot access a shared resource, the "defined(resource_name)" function must be specified in the resource requirement string. For example:

```
% bsub -R "defined(avail_scratch) && avail_scratch > 100 && swap > 100" myjob
```

would exclude any hosts which cannot access the scratch resource. The LSF administrator configures which hosts do and do not have access to a particular shared resource.

Shared resources can also work together with the resource reservation mechanism of LSF Batch to prevent over-committing resources when scheduling. To indicate that a shared resource is to be reserved while a job is running, specify the resource name in the 'rusage' section of the resource requirement string. For example:

```
% bsub -R "select[defined(verilog_lic)] rusage[verilog_lic=1]" myjob
```

would schedule the job on a host when there is verilog license available. The license will be reserved by the job after it is scheduled, until it completes.

Configuring Resource Requirements

Some applications require resources other than the default. LSF can store resource requirements for specific applications so that the application automatically runs with the correct resources. For frequently used commands and software packages, the LSF administrator can set up cluster-wide resource requirements available to all users in the cluster. See the *LSF Batch Administrator's Guide* for more information.

You may have applications that you need to control yourself. Perhaps your administrator did not set them up for load sharing for all users, or you need a non-standard setup. You can use LSF commands to find out resource names available in your system, and tell LSF about the needs of your applications. LSF stores the resource requirements for you from then on.

Remote Task List File

A *task* is a UNIX or NT command or a user-created executable program; the terms *application* or *job* are also used to refer to tasks.

The resource requirements of applications are stored in the *remote task list file*. When you run a job through LSF, LSF automatically picks up the job's default resource requirement string from the remote task list files, unless you explicitly override the default by specifying the resource requirement string on the command line.

There are three sets of task list files: the system-wide default file `lsf.task`, the cluster default file `lsf.task.cluster`, and the user file `$HOME/.lsftask`. The system and cluster default files apply to all users. The user file specifies the tasks to be added to or removed from the system lists for your jobs. Resource requirements specified in your user file override those in the system lists.

Managing Your Task List

The `lsrtasks` command inspects and modifies the remote task list. Invoking `lsrtasks` commands with no arguments displays the resource requirements of tasks in the remote list, separated from the task name by `'/'`.

4 Resources

```
% lsrtasks
```

```
cc/cpu          cfd3d/type == SG1 && cpu compressdir/cpu:mem
f77/cpu         verilog/cpu && cadence  compress/cpu
dsim/type == any hspice/cpu && cadence  nas/swp > 200 && cpu
compress/-:cpu:mem epi/hpux11 sparc      regression/cpu
cc/type == local  synopsys/swp >150 && cpu
```

You can specify resource requirements when tasks are added to the user's remote task list. If the task to be added is already in the list, its resource requirements are replaced.

```
% lsrtasks + myjob/swap>=100 && cpu
```

This adds `myjob` to the remote tasks list with its resource requirements.

Using Resource Requirements

Most LSF commands accept a `-R resreq` argument to specify resource requirements. The exact behaviour depends on the command; for example, specifying a resource requirement for the `lsload` command displays the load levels for all hosts that have the requested resources.

Specifying resource requirements for the `lsrun` command causes LSF to select the best host out of the set of hosts that have the requested resources. The `-R resreq` option overrides any resource requirements specified in the remote task list. For an example of the `lsrun` command with the `-R resreq` option see '*Running Remote Jobs with lsrun*' on page 140.

5. Using LSF Batch

LSF Batch is a distributed batch system for clusters of UNIX and Windows NT computers. With LSF Batch, you can use a heterogeneous network of computers as a single system. All batch jobs go through a consistent interface, independent of the resources they need or the hosts they run on.

LSF Batch has the same view of cluster and master host as the LSF Base, although LSF Batch may only use some of the hosts in the cluster as servers. The slave batch daemon, `sbatchd`, runs on every host that the LSF administrator configures as an LSF Batch server. The master batch daemon, `mbatchd`, always runs on the same host as the master LIM. See *'Finding the Master' on page 25* for more information on the master LIM.

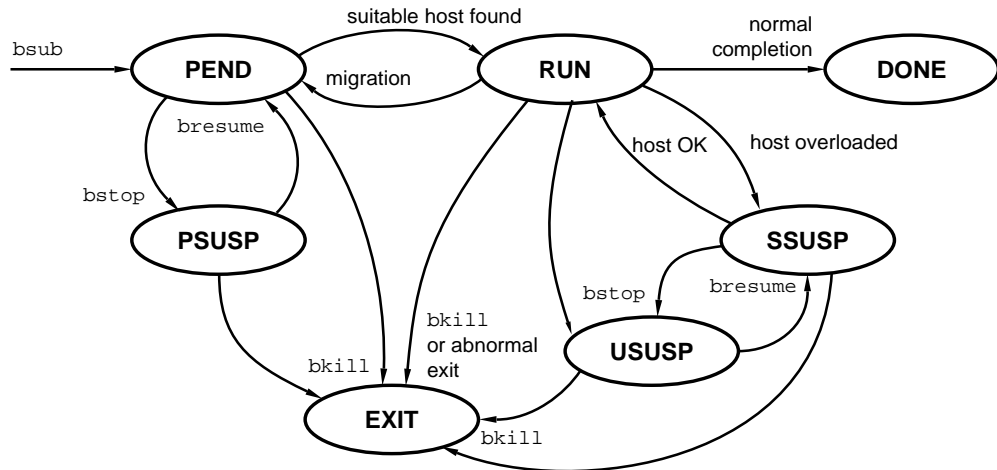
This chapter provides important background information on how LSF Batch works and describes the commands that give information about your LSF Batch system. Topics include:

- the states of an LSF Batch job
- job scheduling policy
- listing batch queues
- choosing the right queue for your job
- how LSF Batch decides when and where to run your job
- listing batch server hosts
- user groups and host groups

Batch Jobs

Each LSF Batch job goes through a series of state transitions until it eventually completes its task, crashes or is terminated. *Figure 9* shows the possible states of a job during its life cycle.

Figure 9. Batch Job States



Many jobs enter only three states:

PEND - waiting in the queue

RUN - dispatched to a host and running

DONE - terminated normally

A job remains pending until all conditions for its execution are met. The conditions may include:

- Start time specified by the user when the job is submitted
- Load conditions on qualified hosts

-
- Time windows during which the job's queue can dispatch jobs and qualified hosts accept jobs
 - Job limits imposed by the configured policy for each user, queue, and host
 - Relative priority to other users and jobs
 - Availability of the specified resources

A job may terminate abnormally for various reasons. Job termination may happen from any state. An abnormally terminated job goes into `EXIT` state. The situations where a job terminates abnormally include:

- The job is cancelled by its owner or the LSF administrator while pending, or after being dispatched
- The job is not able to be dispatched before it reaches its termination deadline and thus is aborted by LSF Batch
- The job fails to start successfully. For example, the wrong executable is specified by the user when the job is submitted
- The job crashes during execution

Jobs may also be suspended at any time. A job can be suspended by its owner, by the LSF administrator or by the LSF Batch system. There are three different states for suspended jobs:

PSUSP - suspended by its owner or the LSF administrator while in `PEND` state

USUSP - suspended by its owner or the LSF administrator after being dispatched

SSUSP - suspended by the LSF Batch system after being dispatched

After a job has been dispatched and started on a host, it is suspended by the LSF Batch system if the load on the execution host or hosts becomes too high. In such a case, batch jobs could be interfering among themselves or could be interfering with interactive jobs. In either case, some jobs should be suspended to maximize host performance or to guarantee interactive response time. LSF Batch suspends jobs according to their priority.

When a host is busy, LSF Batch suspends lower priority jobs first unless the scheduling policy associated with the job dictates otherwise. A job may also be suspended by the system if the job queue has a time window and the current time goes outside the time window.

A system suspended job can later be resumed by LSF Batch if the load condition on the execution host becomes good enough or when the closed time window of the queue opens again.

Fairshare Scheduling Policy

The default First-Come-First-Served (FCFS) job scheduling is often insufficient for an environment with competing users. Fairshare scheduling is an alternative to the default FCFS scheduling. Fairshare scheduling divides the processing power of the LSF cluster among users and groups to provide fair access to resources. Fairshare is not necessarily equal share. Your cluster administrator can configure shares for users or groups to achieve controlled accesses to resources.

Your LSF cluster administrator defines fairshare policies by assigning shares to users or groups. The special names `others` and `default` can also be assigned shares.

The name `others` is a virtual group referring to all other users not explicitly listed in the share parameter. For example, `product` group may be assigned 100 shares, while all `others` together assigned 10 shares.

The name `default` is a virtual user referring to each of the other users not explicitly configured in the share parameter. For example, if the `product` group is assigned 100 shares and `default` user assigned 10 shares, then every user not belonging to the `product` group will have 10 shares, as if their user names were explicitly listed in the share parameter. As a special case, if `default` is the only user name in the share parameter, it implements the equal share policy.

LSF Batch uses an account to maintain information about shares and resource consumption of every user or group. A dynamic priority is calculated for each user or group according to configured shares, CPU time consumed (CPU time used for fairshare is not normalized by the host CPU speed factors) for the past `HIST_HOURS` hours (see '*Configuration Parameters*' on page 85), number of jobs currently running, and

the total elapsed time of jobs. This dynamic priority is then used to decide which user's or group's jobs should be dispatched first. If some users or groups have used less than their fairshare of the resources, their pending jobs (if any) are scheduled next, jumping ahead of jobs of other users.

Note

The CPU time used for host partition scheduling is not normalized by the host CPU speed factors.

LSF Batch provides three different varieties of fairshare configuration. These are queue level fairshare, host partition fairshare, and hierarchical fairshare.

Host Partition Fairshare Scheduling

Host partition fairshare scheduling allows sharing policy to be defined for a group of hosts, rather than in a queue. A host partition specifies a group of hosts together with share allocations among the users or groups. A special host name `all` can be used to refer to all hosts used by LSF Batch.

Note that only users or groups who are configured to use the host partition can run jobs on these hosts.

Fairshare defined by host partition applies to all queues that run jobs on these hosts.

To find out what host partitions are configured in your cluster, run 'bupart' command.

Note

Host partition fairshare is an alternative to queue level fairshare scheduling. You cannot use both in the same LSF cluster.

Queue-Level Fairshare Scheduling

Fairshare policy can be defined at the queue level to allow different policies to be applied for different queues. Queue-level fairshare handles resource contention among user jobs within the same queue.

To find out if a queue has fairshare defined, run the `bqueues -l` command. Your queue has fairshare defined if you see the parameter "USER_SHARES" in the output of the above command.

5 Using LSF Batch

Note

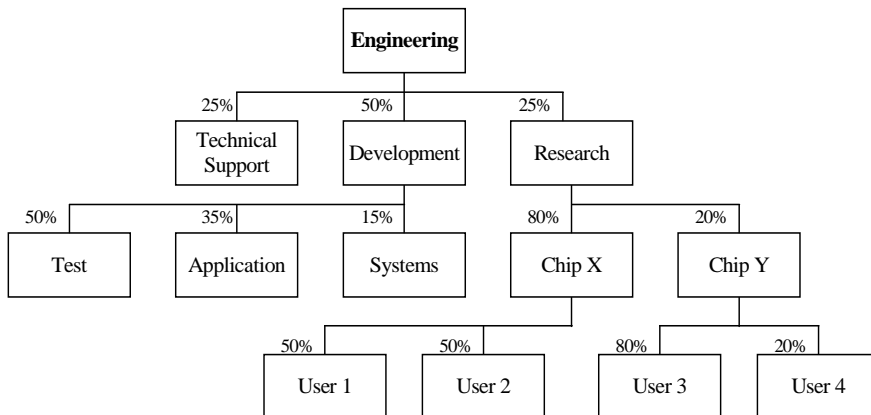
Queue level fairshare scheduling is an alternative to host partition fairshare scheduling. You cannot use both in the same LSF cluster.

Hierarchical Fairshare

When assigning shares in the fairshare queue or host partition to a user group, each member of the group can be given the same share, or all members are collectively given the share. When the share is collectively assigned, the share each member receives depends on the size of the group and the number of jobs submitted by its members.

With large user groups, it is desirable that the shares assigned to a group are subdivided among subgroups. The shares may be further partitioned within subgroups to create a hierarchical share assignment. *Figure 10* gives an example of how an engineering department might want to configure sharing among several groups.

Figure 10. Sample Fairsharing Configuration



The situation pictured in *Figure 10* is a share tree in which users are organized into hierarchical groups. Shares are assigned to users or groups at each level in the hierarchy. In the above example, the Development group will get the largest share (50%) of the resources in the event of contention. Shares assigned to the Development group can be further divided among the Systems, Application and Test groups which

receive 50%, 35%, and 15%, respectively. At the lowest level, individual users may be allocated shares of the immediate group they belong to.

Each node in the share tree represents either a group account or a user account. A user account corresponds to an individual user who runs jobs while a group account allows for assigning shares collectively to a group and subdividing the shares amongst its members. Note that user accounts are leaf nodes in the share tree while group accounts are always non-leaf nodes. The resource consumption of a group account is the total of the consumption of all users accounts defined recursively under that group. By assigning shares to groups, the administrator can control the rate of allocation of resources to all members of the group.

LSF Batch implements hierarchical fairshare in two steps. First, define hierarchical share distribution by defining hierarchical user groups as discussed above. Second, use the hierarchical share distribution in queue-level fairshare or host partition fairshare definitions. When a fairshare policy uses a group name that represents a hierarchical share distribution, it allocates resources according to the share distribution as if the hierarchy were defined inside the policy.

Hierarchical share distribution information can be displayed by the `bugroup` command with `-l` option. See *‘Viewing Hierarchical Share Information’ on page 82* for more information.

Each host partition or queue is considered a share provider and may specify its own fairshare hierarchy for controlling the allocation of resources to its users. For example, a user or group may have large shares in one queue but a small share in the other. When a share provider selects a job to run it searches the share tree from the top and picks the node at each level with the highest priority until a leaf node corresponding to a user is encountered. A job from that user is selected and dispatched if a suitable host is found and that user’s priority together with that of its parent groups is updated. As a user or group account dispatches jobs, its priority will decrease, giving other users or groups a chance to access the resources.

As a user you can belong to one or more of the groups in the share tree. There will be a separate user account for each group you belong to. The priority of your jobs will be affected by the group’s share assignment. When there is contention for resources among the groups, the system will favour those groups with a larger share. Users belonging to multiple groups can specify the share account that will be used to determine the priority of each job. Note that a given user may not have an account

under a particular group. In the previous example, 'User3' does not have an account under the 'ChipX' group.

Other Scheduling Policies

This section discusses other LSF Batch scheduling policies. All these policies can co-exist in the same system.

Preemptive Scheduling

When LSF Batch schedules jobs, those in higher priority queues are considered first. Jobs in lower priority queues are only started if all higher priority jobs are waiting for specified resources, hosts, starting times, or other constraints.

When a high priority job is ready to run, all the LSF Batch server hosts may already be running lower priority jobs. The high priority job ends up waiting for the low priority jobs to finish. If the low priority jobs take a long time to complete, the higher priority jobs may be blocked for an unacceptably long time.

LSF solves this problem by allowing preemptive scheduling within LSF Batch queues. Jobs pending in a preemptive queue can preempt lower priority jobs on a host by suspending them and starting the higher priority jobs on the host.

A queue can also be defined as preemptable. In this case, jobs in higher priority queues can preempt jobs in the preemptable queue even if the higher priority queues are not specified as preemptive.

Note

When the preemptive scheduling policy is used, jobs in preemptive queues may violate the user or host job slot limits. However, LSF Batch ensures that the total number of slots used by running jobs (excluding jobs that are suspended) does not exceed the job slot limits. This is done by suspending lower priority jobs.

Exclusive Scheduling

Some queues accept exclusive jobs. A job can run exclusively only if it is submitted with the `-x` option to the `bsub` command specifying a queue that is configured to accept exclusive jobs. An exclusive job runs by itself on a host — it is dispatched only to a host with no other batch jobs running and LSF does not send any other jobs to the host until the exclusive job completes.

Once an exclusive job is started on a host, the LSF Batch system locks that host out of load sharing by sending a request to the underlying LSF to change the host's status to `lockU`. The host is no longer available for load sharing by any other task (either interactive or batch) until the exclusive job finishes.

Processor Reservation

The scheduling of parallel jobs supports the notion of processor reservation. Parallel jobs requiring a large number of processors can often not be started if there are many lower priority sequential jobs in the system. There may not be enough resources at any one instant to satisfy a large parallel job, but there may be enough to allow a sequential job to be started. With the processor reservation feature the problem of starvation of parallel jobs can be reduced.

When a parallel job cannot be dispatched because there aren't enough execution slots to satisfy its minimum processor requirements, the currently available slots will be reserved for the job. These reserved job slots are accumulated until there are enough available to start the job. When a slot is reserved for a job it is unavailable to any other job.

To use this feature, a queue must have processor reservation policy enabled through the `SLOT_RESERVE` parameter (see '*Processor Reservation for Parallel Jobs*' on page 211 of the *LSF Batch Administrator's Guide*). To avoid deadlock situations, the period of reservation is specified through the `MAX_RESERVE_TIME` parameter. The system will accumulate reserved slots for a job until `MAX_RESERVE_TIME` minutes and if an insufficient number have been accumulated, all slots are freed and made available to other jobs. The `MAX_RESERVE_TIME` parameter takes effect from the start of the first reservation for a job and a job can go through multiple reservation cycles before it accumulates enough slots to be actually started.

Reserved slots can be displayed with the `bjobs` command. The number of reserved slots can be displayed with the `bqueues`, `bhosts`, `bhpart`, and `busers` commands. Look in the `RSV` column.

Backfill Scheduling

Processor reservation ensures that large parallel jobs will not suffer processor starvation. However, in a heavily loaded LSF Batch system with jobs requiring a varying number of processors, a large number of parallel jobs submitted earlier may keep reserving processors. In such cases, FCFS discipline imposes long average wait times on each job, and thereby degrades the system's utilization as many available processor slots are reserved but not used. Backfill policy allows jobs requiring fewer processors and running for shorter periods of time to use the processors reserved by the larger parallel jobs, if these smaller jobs will not delay the start of any of the large parallel jobs.

Since jobs that are backfilled cannot delay the start of that jobs that reserved the job slots, backfilled jobs will not be preempted in any case.

Scheduling Parameters

Scheduling parameters specify the load conditions under which pending jobs are dispatched, running jobs are suspended, and suspended jobs are resumed. These parameters are configured by the LSF administrator in a variety of ways.

Load Thresholds

Load thresholds can be configured by your LSF administrator to schedule jobs in queues. There are two possible types of load thresholds: `loadSched` and `loadStop`. Each load threshold specifies a load index value. A `loadSched` threshold is the scheduling threshold which determines the load condition for dispatching pending jobs. If a host's load is beyond any defined `loadSched`, a job will not be started on the host. This threshold is also used as the condition for resuming suspended jobs. A `loadStop` threshold is the suspending condition that determines when running jobs should be suspended.

Thresholds can be configured for each queue, for each host, or a combination of both. To schedule a job on a host, the load levels on that host must satisfy both the thresholds configured for that host and the thresholds for the queue from which the job is being dispatched.

The value of a load index may either increase or decrease with load, depending on the meaning of the specific load index. Therefore, when comparing the host load conditions with the threshold values, you need to use either greater than (>) or less than (<), depending on the load index.

When jobs are running on a host, LSF Batch periodically checks the load levels on that host. If any load index exceeds the corresponding per-host or per-queue suspending threshold for a job, LSF Batch suspends the job. The job remains suspended until the load levels satisfy the scheduling thresholds.

To find out what parameters are configured for your cluster, see *'Detailed Queue Information'* on page 69 and *'Batch Hosts'* on page 79.

Scheduling Conditions

Scheduling conditions are a more general way for specifying job dispatching conditions at the queue level. Three parameters, `RES_REQ`, `STOP_COND` and `RESUME_COND`, can be specified in the definition of a queue. These parameters take resource requirement strings as values (see *'Resource Requirement Strings'* on page 46 for more details) which results in a more flexible specification of conditions than load threshold.

The resource requirement conditions for dispatching a job to a host can be specified through the queue level `RES_REQ` parameter (see *'Queue-Level Resource Requirement'* on page 213 of the *LSF Batch Administrator's Guide* for further details). This parameter provides an alternative for 'loadshare' as described in *'Load Thresholds'* on page 64.

You can also specify the resource requirements for your job using the `-R` option to the `bsub` command. If you specify resource requirements that are already defined in the queue, the host must satisfy both requirements to be eligible for running the job. In some cases, the queue specification sets an upper or lower bound on a resource. If you attempt to exceed that bound, your job will be rejected.

The condition for suspending a job can be specified using the queue level `STOP_COND` parameter. It is defined by a resource requirement string (see *'Suspending Condition'* on

page 215 of the *LSF Batch Administrator's Guide*). The stopping condition can only be specified in the queue. This parameter provides similar but more flexible function for `loadStop` as described in *'Load Thresholds'* on page 64.

The resource requirement conditions that must be satisfied on a host before a suspended job can be resumed is specified using the queue level `RESUME_COND` parameter (for more detail see *'Resume Condition'* on page 215 of the *LSF Batch Administrator's Guide*). The resume condition can only be specified in the queue.

To find out details about the parameters of your cluster, see *'Detailed Queue Information'* on page 69 and *'Batch Hosts'* on page 79.

Time Windows for Queues and Hosts

Separate time windows can be defined to control when jobs can be dispatched and when they are to be suspended.

Run Windows

Run windows are time windows during which jobs are allowed to run. When the windows are closed, running jobs are suspended and no new jobs are dispatched. The default is no restriction, or always open. Run windows can only be defined for queues (see *'Detailed Queue Information'* on page 69).

Note

*These windows are only applicable to batch jobs. Interactive jobs scheduled by the Load Information Manager (LIM) of LSF are controlled by another set of run windows (see *'Listing Hosts'* on page 27).*

Dispatch Windows

Dispatch windows are time windows during which jobs are allowed to be started. However, dispatch windows have no effect on jobs that have already started. This means that jobs are allowed to run outside the dispatch windows, but no new jobs will be started. The default is no restriction, or always open. Note that no jobs are allowed to start when the run windows are closed. Dispatch windows can be defined for both

queues (see ‘*Detailed Queue Information*’ on page 69) and batch server hosts (see ‘*Batch Hosts*’ on page 79).

Batch Queues

Batch queues represent different job scheduling and control policies. All jobs submitted to the same queue share the same scheduling and control policy. Batch queues do not correspond to individual hosts; each job queue can use all server hosts in the cluster, or a configured subset of the server hosts.

The LSF administrator can configure job queues to control resource accesses by different users and types of application. Users select the job queue that best fits each job.

Finding Out What Queues Are Available

The `bqueues` command lists the available LSF Batch queues.

% **bqueues**

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
interactive	400	Open:Active	-	-	-	-	2	0	2	0
priority	43	Open:Active	-	-	-	-	16	4	11	1
night	40	Open:Inactive	-	-	-	-	4	4	0	0
short	35	Open:Active	-	-	-	-	6	1	5	0
license	33	Open:Active	-	-	-	-	0	0	0	0
normal	30	Open:Active	-	-	-	-	0	0	0	0
idle	20	Open:Active	-	-	-	-	6	3	1	2

The `PRIO` column gives the priority of the queue. The bigger the value, the higher the priority. Queue priorities are used by LSF Batch for job scheduling and control. Jobs from higher priority queues are dispatched first. Jobs from lower priority queues are suspended first when hosts are overloaded.

The `STATUS` column shows the queue status. A queue accepts new jobs only if it is *open* and dispatches jobs only if it is *active*. A queue can be opened or closed only by the LSF administrator. Jobs submitted to a queue that is later closed are still dispatched as long

5 Using LSF Batch

as the queue is active. A queue can be made active or inactive either by the LSF administrator or by the run and dispatch windows of the queue.

The `MAX` column shows the limit on the number of jobs dispatched from this queue at one time. This limit prevents jobs from a single queue from using too many hosts in a cluster at one time.

The `JL/U` column shows the limit on the number of jobs dispatched at one time from this queue for each user. This prevents a single user from occupying too many hosts in a cluster while other users' jobs are waiting in the queue.

The `JL/P` column shows the limit on the number of jobs from this queue dispatched to each processor. This prevents a single queue from occupying too many of the resources on a host.

The `JL/H` column shows the maximum number of job slots a host can allocate for this queue. This limit controls the number of job slots for the queue on each host, regardless of the type of host: uniprocessor or multiprocessor.

The `NJOBS` column shows the total number of job slots required by all jobs in the queue, including jobs that have not been dispatched and jobs that have been dispatched but have not finished.

Note

A parallel job with N components would require N job slots.

The `PEND` column shows the number of job slots needed by pending jobs in this queue.

The `RUN` column shows the number of job slots used by running jobs in this queue.

The `SUSP` column shows the number of job slots required by suspended jobs in this queue.

Detailed Queue Information

The `-l` option to the `bqueues` command displays the complete status and configuration for each queue. You can specify queue names on the command line to select specific queues:

```
% bqueues -l normal
```

```
QUEUE: normal
```

```
--
```

For normal low priority jobs, running only if hosts are lightly loaded. This is the default queue.

PARAMETERS/STATISTICS

PRIO	NICE	STATUS	MAX	JL/U	JL/P	NJOBS	PEND	RUN	SSUSP	USUSP
40	20	Open:Active	100	50	11	1	1	0	0	0

Migration threshold is 30 min.

CPULIMIT

20 min of IBM350

RUNLIMIT

342800 min of IBM350

FILELIMIT

20000 K

DATALIMIT

20000 K

STACKLIMIT

2048 K

CORELIMIT

20000 K

MEMLIMIT

5000 K

PROCLIMIT

3

SCHEDULING PARAMETERS

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	0.7	1.0	0.2	4.0	50	-	-	-	-	-
loadStop	-	1.5	2.5	-	8.0	240	-	-	-	-	-

SCHEDULING POLICIES: FAIRSHARE PREEMPTIVE PREEMPTABLE EXCLUSIVE

USER_SHARES: [groupA, 70] [groupB, 15] [default, 1]

DEFAULT HOST SPECIFICATION : IBM350

RUN_WINDOWS: 2:40-23:00 23:30-1:30

DISPATCH_WINDOWS: 1:00-23:50

USERS: groupA/ groupB/ user5

HOSTS: hostA, hostD, hostB

5 Using LSF Batch

```
ADMINISTRATORS:  user7
PRE_EXEC:  /tmp/apex_pre.x > /tmp/preexec.log 2>&1
POST_EXEC:  /tmp/apex_post.x > /tmp/postexec.log 2>&1
REQUEUE_EXIT_VALUES:  45
```

The `bqueues -l` command only displays fields that apply to the queue. Any field that is not displayed has a default value that does not affect job scheduling or execution. In addition to the fields displayed by the default `bqueues` command, the fields that may be displayed are:

DESCRIPTION

A description of the typical use of the queue.

Default queue indication

Indicates that this is the default queue.

SSUSP

The number of job slots required by jobs suspended by the system because of load levels or run windows.

USUSP

The number of jobs slots required by jobs suspended by the user or the LSF administrator.

RSV

The numbers of job slots in the queue that are reserved by LSF Batch for pending jobs.

Migration threshold

The time that a job dispatched from this queue will remain suspended by the system before LSF Batch attempts to migrate the job to another host.

CPULIMIT

The maximum CPU time a job can use, in minutes relative to the CPU factor of the named host. `CPULIMIT` is scaled by the CPU factor of the execution host so that jobs are allowed more time on slower hosts.

When the job-level `CPULIMIT` is reached, the system sends `SIGXCPU` to all processes belonging to the job.

RUNLIMIT

The maximum wall clock time a process can use, in minutes. **RUNLIMIT** is scaled by the CPU factor of the execution host. When a job has been in the **RUN** state for a total of **RUNLIMIT** minutes, LSF Batch sends a **SIGUSR2** signal to the job. If the job does not exit within 10 minutes, LSF Batch sends a **SIGKILL** signal to kill the job.

FILELIMIT

The maximum file size a process can create, in kilobytes. This limit is enforced by the UNIX **setrlimit** system call if it supports the **RLIMIT_FSIZE** option, or the **ulimit** system call if it supports the **UL_SETFSIZE** option.

DATALIMIT

The maximum size of the data segment of a process, in kilobytes. This restricts the amount of memory a process can allocate. **DATALIMIT** is enforced by the **setrlimit** system call if it supports the **RLIMIT_DATA** option, and unsupported otherwise.

STACKLIMIT

The maximum size of the stack segment of a process, in kilobytes. This restricts the amount of memory a process can use for local variables or recursive function calls. **STACKLIMIT** is enforced by the **setrlimit** system call if it supports the **RLIMIT_STACK** option.

CORELIMIT

The maximum size of a core file, in kilobytes. This limit is enforced by the **setrlimit** system call if it supports the **RLIMIT_CORE** option.

MEMLIMIT

The maximum running set size (RSS) of a process, in kilobytes. If a process uses more than **MEMLIMIT** kilobytes of memory, its priority is reduced so that other processes are more likely to be paged in to available memory. This limit is enforced by the **setrlimit** system call if it supports the **RLIMIT_RSS** option.

PROCLIMIT

The maximum number of processors allocated to a job. Jobs requesting more processors than the queue's **PROCLIMIT** are rejected.

5 Using LSF Batch

PROCESSLIMIT

The maximum number of concurrent processes allocated to a job. If PROCESSLIMIT is reached, the system sends the following signals in sequence to all processes belonging to the job: SIGINT, SIGTERM, and SIGKILL.

SWAPLIMIT

The swap space limit that a job may use. If SWAPLIMIT is reached, the system sends the following signals in sequence to all processes in the job: SIGINT, SIGTERM, and SIGKILL.

loadSched

The load thresholds LSF Batch uses to determine whether a pending job in this queue can be dispatched to a host, and to determine when a suspended job can be resumed. The load indices are explained in *'Load Indices' on page 37*.

loadStop

The load thresholds LSF Batch uses to determine when to suspend a running batch job in this queue.

SCHEDULING POLICIES

Scheduling policies of the queue. Optionally, one or more of the following policies may be configured:

FAIRSHARE

Jobs in this queue are scheduled based on a fairshare policy. In general, a job will be dispatched before other jobs in this queue if the job's owner has more shares (see USER_SHARES below), fewer running jobs, and has used less CPU time in the recent past, and the job has waited longer. If all the users have the same shares, jobs in this queue are scheduled in a round-robin fashion.

If the fairshare policy is not specified, jobs in this queue are scheduled based on the conventional first-come-first-served (FCFS) policy. That is, jobs are dispatched in the order they were submitted.

PREEMPTIVE

Jobs in this queue may preempt running jobs from lower priority queues. That is, jobs in this queue may still be able to start even though the job limit of a host or a user has been reached, as long as some of the job slots defined by the job limit are taken by jobs from those queues whose priorities are lower than the priority of this queue. Jobs from lower priority queues will be suspended to ensure that the running jobs (excluding suspended jobs) are within the

corresponding job limit. If the preemptive policy is not specified, the default is not to preempt any job.

PREEMPTABLE

Jobs in this queue may be preempted by jobs in higher priority queues, even if the higher priority queues are not specified as preemptive.

EXCLUSIVE

Jobs dispatched from this queue can run exclusively on a host if the user so specifies at job submission time (see ‘*Other bsub Options*’ on page 112). Exclusive execution means that the job is sent to a host with no other batch jobs running there, and no further job—batch or interactive—will be dispatched to that host while the job is running. The default is not to allow exclusive jobs.

BACKFILL

Parallel jobs can reserve job slots on hosts so that they are not prevented from executing if they are competing with jobs requiring fewer processors (as specified via `bsub -n`). This maximum slot reservation time controls how long, in seconds, a slot is reserved for a job. The backfill policy allows a site to make use of the reserved slots for short jobs without delaying the starting time of the parallel job doing the reserving.

The run limit of currently started jobs is used to compute the estimated start time of a job when backfilling is enabled. A job can backfill the reserved slots of another job if it will finish, based on its run limit, before the estimated start time of the backfilled job. Jobs in a backfill queue can backfill any jobs which are reserving slots. If backfilling is enabled, the estimated start time of a job can be viewed using `bjobs -l`. LSF Batch provides support for Backfill at the queue level.

USER_SHARES

A list of [*username*, *share*] pairs. *username* is either a user name or a user group name. *share* is the number of shares of resources assigned to the user or user group. A party will get a portion of the resources proportional to the party’s share divided by the sum of the shares of all parties specified in this queue.

DEFAULT HOST SPECIFICATION

A host name or host model name. The appropriate CPU scaling factor of the host or host model (see `lsinfo(1)`) is used to adjust the actual CPU time limit at the execution host (see `CPULIMIT` above). This specification overrides the

5 Using LSF Batch

system default `DEFAULT_HOST_SPEC` (see ‘*Configuration Parameters*’ on page 85).

RUN_WINDOWS

One or more run windows in a week during which jobs in this queue may execute. When a queue is out of its window or windows, no job in this queue will be dispatched. In addition, when the end of a run window is reached, any running jobs from this queue are suspended until the beginning of the next run window, when they are resumed. The default is no restriction, or always open.

A window is displayed in the format of *begin_time-end_time*. Time is specified in the format of *[day:]hour[:minute]*, where all fields are numbers in their respective legal ranges: 0(Sunday)-6 for day, 0-23 for hour, and 0-59 for minute. The default value for minute is 0 (on the hour). The default value for day is every day of the week. The *begin_time* and *end_time* of a window are separated by ``-'`, with no blank characters (SPACE or TAB) in between. Both *begin_time* and *end_time* must be present for a window. Windows are separated by blank characters. If only the character ``-'` is displayed, the windows are always open.

DISPATCH_WINDOWS

One or more dispatch windows in a week during which jobs in this queue may be dispatched to run. When a queue is out of its windows, no job in this queue can be dispatched. Jobs already dispatched are not affected by the dispatch windows. The default is no restriction, or always open. Dispatch windows are displayed in the same format as run windows (see `RUN_WINDOWS` above).

USERS

The list of users allowed to submit jobs to this queue.

HOSTS

The list of hosts to which this queue can dispatch jobs.

NQS_DESTINATION_QUEUES

The list of NQS queues to which this queue can dispatch jobs.

ADMINISTRATORS

A list of administrators of the queue. The users whose names are specified here are allowed to operate on the jobs in the queue and on the queue itself.

JOB_STARTER

An executable file that runs immediately prior to the batch job, taking the batch job file as an input argument. All jobs submitted to the queue are run via the job starter, which is generally used to create a specific execution environment before processing the jobs themselves.

PRE_EXEC

Queue's pre-execution command. This command is executed before the real batch job is run on the execution host (or on the first host selected for a parallel batch job).

POST_EXEC

Queue's post-execution command. This command is executed on the execution host when a job terminates.

REQUEUE_EXIT_VALUES

Jobs that exit with these values are automatically requeued.

RES_REQ

Resource requirements of the queue. Only the hosts that satisfied this resource requirement can be used by the queue.

RESUME_COND

The condition(s) that must be satisfied to resume a suspended job on a host.

STOP_COND

The condition(s) which determine whether a job running on a host should be suspended.

Note that some parameters are displayed only if they are defined.

Automatic Queue Selection

When more than one batch queue is available, you need to decide which queue to use. If you submit a job without specifying a queue name, the LSF Batch system automatically chooses a suitable queue for the job from the candidate default queues, based on the requirements of the job.

Specifying Default Queues

LSF Batch has default queues. The `bparams` command displays them:

```
% bparams
Default Queues: normal
...
```

The user can override this list by defining the environment variable `LSB_DEFAULTQUEUE`.

Queue Selection Mechanism

Although simple to use, automatic queue selection may not behave as expected, if you do not choose your candidate queues properly. The criteria LSF Batch uses for selecting a suitable queue are as follows:

- User access restriction. Queues that do not allow this user to submit jobs are discarded
- Host restriction. If the job explicitly specifies a list of hosts on which the job can be run, then the selected queue must be configured to send jobs to all hosts in the list
- Queue status. Closed queues are not considered
- Exclusive execution restriction. If the job requires exclusive execution, then queues that are not configured to accept exclusive jobs are discarded
- Job's requested resources. These must be within the resource limits of the selected queue

If multiple queues satisfy the above requirements, then the first queue listed in the candidate queues (as defined by `DEFAULT_QUEUE` or `LSB_DEFAULTQUEUE`) that satisfies the requirements is selected.

Choosing a Queue

The default queues are normally suitable to run most jobs for most users, but they may have a very low priority or restrictive execution conditions to minimize interference with other jobs. If automatic queue selection is not satisfactory, you should choose the most suitable queue for each job.

The factors affecting your decision are user access restrictions, size of the job, resource limits of the queue, scheduling priority of the queue, active time windows of the queue, hosts used by the queue, the scheduling load conditions, and the queue description displayed by the `bqueues -l` command.

The `-u user_name` option specifies a user or user group so that `bqueues` displays only the queues that accept jobs from these users.

The `-m host_name` option allows users to specify a host name or host group name so that `bqueues` displays only the queues that use these hosts to run jobs.

You must also be sure that the queue is enabled.

The following examples are based on the queues defined in the default LSF configuration. Your LSF administrator may have configured different queues.

To run a job during off hours because the job generates very high load to both the file server and the network, you can submit it to the night queue; use `bsub -q night`.

If you have an urgent job to run, you may want to submit it to the priority queue; use `bsub -q priority`.

If you want to use hosts owned by others and you do not want to bother the owners, you may want to run your low priority jobs on the idle queue so that as soon as the owner comes back, your jobs get suspended.

If you are running small jobs and do not want to wait too long to get the results, you can submit jobs to the short queue to be dispatched with higher priority. Make sure your jobs are short enough that they are not killed for exceeding the CPU time limit of the queue (check the resource limits of the queue, if any).

5 Using LSF Batch

If your job requires a specific execution environment, you may need to submit it to a queue that has a particular job starter defined. Because only your system administrator is able to specify a queue-level job starter as part of the queue definition, you should check with him for more information. See ‘*Queue-Level Job Starters*’ on page 129 of the *LSF Batch Administrator’s Guide* for information on queue-level job starters.

Batch Users

The `busers` command displays the maximum number of jobs a user or group may execute on a single processor, the maximum number of job slots a user or group may use in the cluster, the total number of job slots required by all submitted jobs of the user, and the number of job slots in the `PEND`, `RUN`, `SSUSP`, and `USUSP` states. If no user is specified, the default is to display information about the user who invokes this command. Here is an example of the output from the `busers` command:

```
% busers all
```

USER/GROUP	JL/P	MAX	NJOBS	PEND	RUN	SSUSP	USUSP	RSV
default	1	12	-	-	-	-	-	-
user9	1	12	34	22	10	2	0	0
groupA	-	100	20	7	11	1	1	0

Note that if the reserved user name `all` is specified, `busers` reports all users who currently have jobs in the system, as well as `default`, which represents a typical user. The purpose of listing `default` in the output is to show the job slot limits (`JL/P` and `MAX`) of a typical user. No other parameters make sense for `default`.

Note

The counters displayed by `busers` treat a parallel job requesting N processors the same as N jobs requesting one processor.

Batch Hosts

LSF Batch uses some (or all) of the hosts in an LSF cluster as execution hosts. The host list is configured by the LSF administrator. The `bhosts` command displays information about these hosts.

% `bhosts`

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
hostA	ok	2	2	0	0	0	0	0
hostD	ok	2	4	2	1	0	0	1
hostB	ok	1	2	2	1	0	1	0

`STATUS` gives the status of the host and the `sbatchd` daemon. If a host is down or the LIM is unreachable, the `STATUS` is `unavail`. If the LIM is reachable but the `sbatchd` is not up, `STATUS` is `unreach`.

`JL/U` is the job slot limit per user. The host will not allocate more than `JL/U` job slots for one user at the same time. `MAX` gives the maximum number of job slots that are allowed on this host. This does not mean that the host has to always allocate this many job slots if there are waiting jobs; the host must also satisfy its configured load conditions to accept more jobs.

The columns `NJOBS`, `RUN`, `SSUSP`, `USUSP`, and `RSV` show the number of job slots used by jobs currently dispatched to the host, running on the host, suspended by the system, suspended by the user, and reserved on the host respectively.

The `-l` option to the `bhosts` command gives all information about each batch server host such as the CPU speed factor and the load threshold values for starting, resuming

5 Using LSF Batch

and suspending jobs. You can also specify host names on the command line to list the information for specific hosts.

```
% bhosts -l hostB
```

```
HOST: hostB
STATUS CPUF JL/U MAX NJOBS RUN SSUSP USUSP RSV DISPATCH_WINDOWS
ok      9    1    2    2      1    0      0    1    2:00-20:30

          r15s  rlm  r15m  ut    pg    io    ls    it    tmp  swp  mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    40   -    -    -    -    -    -
```

Migration threshold is 40 min.
Files are copied at checkpoint.

The DISPATCH_WINDOWS column shows the time windows during which jobs can be started on the host. See *'Detailed Queue Information' on page 69* for a description of the format of the DISPATCH_WINDOWS column. Unlike the queue run windows, jobs are not suspended when the host dispatch windows close. Jobs running when the host dispatch windows close continue running, but no new jobs are started until the windows reopen.

CPUF is the host CPU factor. loadSched and loadStop are the scheduling and suspending thresholds for the host. If a threshold is not defined, the threshold from the queue definition applies. If both the host and the queue define a threshold for a load index, the most restrictive threshold is used.

The migration threshold is the time that a job dispatched to this host can remain suspended by the system before LSF Batch attempts to migrate the job to another host.

If the host's operating system supports checkpoint copy, this is indicated here. With checkpoint copy, the operating system automatically copies all open files to the checkpoint directory when a process is checkpointed. Checkpoint copy is currently supported only on ConvexOS and Cray systems.

User and Host Groups

The LSF administrator can configure user and host groups. The group names act as convenient aliases wherever lists of user or host names can be specified on the command line or in configuration files. The administrator can also limit the total number of running jobs belonging to a user or a group of users. User groups can also be defined to reflect the hierarchical share distribution, as discussed in *'Hierarchical Fairshare'* on page 60.

The `bugroup` and `bmgroup` commands list the configured group names and members for user and host groups respectively.

```
% bugroup acct_users
GROUP_NAME    USERS
acct_users :  user1 user2 user4 group1/
```

Note that if a name is ended by a '/', it is a group.

```
% bmgroup big_servers
GROUP_NAME    HOSTS
big_servers :  hostD hostK
```

Specifying a user or host group to an LSF Batch command is the same as specifying all the user or host names in the group. For example, the command `bsub -m big_servers` specifies that the job may be dispatched to either of the hosts *hostD* or *hostK*. The command `bjobs -l` lists detailed information about the job, including the specified hosts and the load thresholds that apply to the job.

```
% bsub -m big_servers myjob
Job <31556> is submitted to default queue <normal>.
```

5 Using LSF Batch

```
% bjobs -l 31556
```

```
Job Id <31556>, User <user1>, Status <DONE>, Queue <normal>, Comm  
and <hostname>
```

```
Thu Oct 27 01:47:51: Submitted from host <hostA>, CWD <$HOME>,  
Specified Hosts <big_servers>;
```

```
Thu Oct 27 01:47:52: Started on <hostK>;
```

```
Thu Oct 27 01:47:53: Done successfully. The CPU time used is 0.2  
seconds.
```

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	12	-
loadStop	-	-	-	-	55	-	-	-	-	-	-

Viewing Hierarchical Share Information

The hierarchical share distribution can be displayed by the `bugroup` command with `-l` option. The following gives an example of a system consisting of three groups:

```
% bugroup -l
```

```
GROUP_NAME: g0  
USERS:      g1/ g2/  
SHARES:     [g2, 20] [g1, 10]
```

```
GROUP_NAME: g1  
USERS:      user1 user2 user3  
SHARES:     [others, 10] [user3, 4]
```

```
GROUP_NAME: g2  
USERS:      all users  
SHARES:     [user2, 10] [default, 5]
```

For fairsharing to take effect, the group share definitions must be associated with individual share providers (queues or host partitions) in the system. For example, if the above share definition was associated with a host partition consisting of hostA, hostB,

and hostC, the `bhpart` command can display the share distribution information. By default, the command only displays the top level share accounts associated with the partition. Use the `-r` option to recursively display the entire share tree associated with the provider.

```
% bhpart hpartest
```

```
HOST_PARTITION_NAME: hpartest
HOSTS:               hostA hostB hostC
```

```
SHARE_INFO_FOR: hpartest:/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
g0	100	5.440	5	0	0.0	1324

```
% bhpart -r hpartest
```

```
HOST_PARTITION_NAME: hpartest
HOSTS: hopper
```

```
SHARE_INFO_FOR: hpartest/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
g0	100	5.477	5	0	0.0	1324

```
SHARE_INFO_FOR: hpartest/g0/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
g2	20	1.645	3	0	0.0	816
g1	10	1.099	2	0	0.0	508

```
SHARE_INFO_FOR: hpartest/g0/g2/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
user3	10	3.333	0	0	0.0	0
user2	5	1.667	3	0	0.0	0
user1	5	1.667	0	0	0.0	0

```
SHARE_INFO_FOR: hpartest/g0/g1/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
user2	4	1.333	0	0	0.0	0
others	10	1.099	2	0	0.0	508

Note that when displaying the share tree recursively, the output consists of a series of group accounts starting from the share provider. Each group contains the account information of any subgroups or users under that group. The `ACCOUNT_PATH` gives the path name of the group account starting from name of the share provider. Each user account similarly can be identified by a unique path e.g `hpartest/g0/g1/user2`.

The information associated with each account includes the static share assigned to that group or user as well as its dynamic priority. A higher value for the priority indicates that the user's or group's jobs will be considered before those with lower priority at the same level. Priorities for accounts at different levels in the tree should not be compared. Details about the number of started and reserved jobs together with the cpu time and run time used by the accounts previously submitted jobs is displayed. Note that the group account's job counters and cpu time fields are the sum of those for all users or subgroups underneath it.

The `-Y` option of `bqueues` will display a similar share tree for a given fairshare queue.

Queue-Level Job Starters

If you frequently need to submit batch jobs that have to be started in a particular environment or require some type of setup to be performed before they are executed, your system administrator can include a *job starter* function in the definition of a selected queue. In a shell environment, this type of pre-execution setup is often handled by writing the preliminary procedures into a file (referred to as a *wrapper*) that itself contains a call to start the desired job.

In LSF, a queue-level job starter does the work of a wrapper. A job starter is simply a command (or set of commands) which, when included in the queue definition, is run immediately prior to all jobs submitted to the selected queue. The job starter performs its setup or environment functions, then calls the submitted job itself, which can inherit the execution environment created by the job starter. One typical use of this feature is to customize LSF for use with Atria ClearCase environment (see '*Support for Atria ClearCase*' on page 275 of the *LSF Batch Administrator's Guide*).

A queue-level job starter can only be specified by the LSF administrator. You can specify a job starter for your interactive jobs using the `LSF_JOB_STARTER`

environment variable. See ‘*Command-Level Job Starters*’ on page 144 for detailed information.

Queue-level job starters have no effect on interactive jobs, unless the interactive job is submitted to a queue as an interactive batch job (see ‘*Interactive Batch Job Support*’ on page 145 for information on interactive batch jobs).

Configuration Parameters

The `bparams` command reports some generic configuration parameters of the LSF Batch system. These include the default queues, default host or host model for CPU speed scaling, job dispatch interval, job checking interval, job accepting interval, etc. The command can display such information in either short format or long format. The short format summarizes a few key parameters. For example:

```
% bparams
Default Queues:   normal idle
Default Host Specification:  DECAXP
Job Dispatch Interval:  20 seconds
Job Checking Interval:  15 seconds
Job Accepting Interval:  20 seconds
```

The `-l` option to the `bparams` command displays the information in long format, which gives a brief description of each parameter as well as the name of the parameter as it appears in the `lsb.params` file. In addition, the long format lists every parameter defined in the `lsb.params` file. Here is an example of the output from the long format of the `bparams` command:

```
% bparams -l

System default queues for automatic queue selection:
    DEFAULT_QUEUE = normal idle

The interval for dispatching jobs by master batch daemon:
    MBD_SLEEP_TIME = 20 (seconds)

The interval for checking jobs by slave batch daemon:
```

5 Using LSF Batch

```
SBD_SLEEP_TIME = 15 (seconds)
```

The interval for a host to accept two batch jobs subsequently:

```
JOB_ACCEPT_INTERVAL = 1 (* MBD_SLEEP_TIME)
```

The idle time of a host for resuming pg suspended jobs:

```
PG_SUSP_IT = 180 (seconds)
```

The amount of time during which finished jobs are kept in core:

```
CLEAN_PERIOD = 3600 (seconds)
```

The maximum number of finished jobs that are logged in current event file:

```
MAX_JOB_NUM = 2000
```

The maximum number of retries for reaching a slave batch daemon:

```
MAX_SBD_FAIL = 3
```

The number of hours of resource consumption history:

```
HIST_HOURS = 5
```

The default project assigned to jobs.

```
DEFAULT_PROJECT = default
```

User Controlled Account Mapping

By default, LSF assumes a uniform user name space within a cluster. Some sites do not satisfy this assumption. For such sites, LSF provides support for the execution of batch jobs within a cluster with a non-uniform user name space.

You can set up a hidden `.lsfhosts` file in your home directory that tells what accounts to use when you send jobs to remote hosts and which remote users are allowed to run jobs under your local account. This is similar to the `.rhosts` file used by `rcp`, `rlogin` and `rsh`.

The `.lsfhosts` file consists of multiple lines, where each line is of the form:

```
hostname|clustername username [send|recv]
```

A '+' in the *hostname* or *username* field indicates any LSF host or user respectively. The keyword `send` indicates that if you send a job to host *hostname*, then the account *username* should be used. The keyword `recv` indicates that your local account is enabled to run jobs from user *username* on host *hostname*. If neither `send` nor `recv` are specified, then your local account can both send jobs to and receive jobs from the account *username* on *hostname*.

Note

The `clustername` argument is used for the LSF MultiCluster product. See 'Using LSF MultiCluster' on page 187

Lines beginning with '#' are ignored.

Note

The permission on your `.lsfhosts` file must be set to read/write only by the owner. Otherwise, your `.lsfhosts` file is silently ignored.

For example, assume that *hostB* and *hostA* in your cluster do not share the same user name/user ID space. You have an account *user1* on host *hostB* and an account *ruser_1* on host *hostA*. You want to be able to submit jobs from *hostB* to run on *hostA*.

Your `.lsfhosts` files should be set up as follows:

On *hostB*:

```
% cat ~user1/.lsfhosts
hostA ruser_1 send
```

On *hostA*:

```
% cat ~ruser_1/.lsfhosts
hostB user1 recv
```

As another example, assume you have account *user1* on host *hostB* and want to use the `lsfguest` account when sending jobs to be run on host *hostA*. The `lsfguest` account is intended to be used by any user submitting jobs from any LSF host.

5 Using LSF Batch

The `.lsfhosts` files should be set up as follows:

On *hostB*:

```
% cat ~user1/.lsfhosts
hostA lsfguest send
```

On *hostA*:

```
% cat ~lsfguest/.lsfhosts
+ + recv
```

When using account mapping, your job is always started as a login shell so that the start-up files of the user account, under which your job will run, are sourced.

Your `.lsfhosts` file is read at job submission time. Subsequent changes made to this file will not affect the account used to run the job. Jobs submitted after the changes are made will pick up the new entries.

If you attempt to map to an account for which you have no permission, your job is put into PSUSP state. You can modify the `.lsfhosts` file of the execution account to give appropriate permission and resume the job.

Note

The `bpeek` command will not work on a job running under a different user account.

File transfer using the `-f` option to the `bsub` command will not work when running under a different user account unless `rcp(1)` is set up to do the file copying.

6. Submitting Batch Jobs

This chapter describes how to use the `bsub` command. Command options are divided into groups with related functions. Topics covered in this chapter are:

- input to and output from batch jobs
- specifying resource requirements
- restricting the hosts eligible to run a job
- controlling resource usage
- using pre-execution commands to determine when a job can start
- specifying dependencies between batch jobs
- moving files to and from the execution host
- specifying 'start after' and 'finish before' times
- submitting parallel jobs
- other LSF Batch options
- job scripts

The options to the `bsub` command related to job checkpointing and migration are described in '*Checkpointing and Migration*' on page 165.

Input and Output

When a batch job completes or exits, LSF Batch by default sends you a job report by electronic mail. The report includes the standard output (`stdout`) and error output (`stderr`) of the job. The output from `stdout` and `stderr` are merged together in the order printed, as if the job was run interactively. The default standard input (`stdin`) file is the null device.

UNIX The null device is `/dev/null`.

If you want mail sent to another user, use the `-u username` option to the `bsub` command. Mail associated with the job will be sent to the named user instead of to you.

If you do not want output to be sent by mail, you can specify `stdout` and `stderr` files. You can also specify the standard input file if the job needs to read input from `stdin`. For example:

```
% bsub -q night -i job_in -o job_out -e job_err myjob
```

submits `myjob` to the `night` queue. The job reads its input from file `job_in`. Standard output is stored in file `job_out`, and standard error is stored in file `job_err`. If you specify a `-o outfile` argument and do not specify a `-e errfile` argument, the standard output and error are merged and stored in `outfile`.

The output file created by the `-o` option to the `bsub` command normally contains job report information as well as the job output. This information includes the submitting user and host, the execution host, the CPU time (user plus system time) used by the job, and the exit status. If you want to separate the job report information from the job output, use the `-N` option to specify that the job report information should be sent by email.

The output files specified by the `-o` and `-e` options are created on the execution host. See *'Remote File Access' on page 101* for an example of copying the output file back to the submission host if the job executes on a file system that is not shared between the submission and execution hosts.

Resource Requirements

If you need to explicitly specify resource requirements for your job, use the `-R` option to the `bsub` command. For example:

```
% bsub -R "swp > 15 && hpux order[cpu]" myjob
```

runs `myjob` on an HP-UX host that is lightly loaded (CPU utilization) and has at least 15 megabytes of swap memory available. See *'Resource Requirement Strings' on page 46* for a complete discussion of resource requirements.

You do not have to specify resource requirements every time you submit a job. The LSF administrator may have already configured the resource requirements for your jobs, or you can put your executable name together with its resource requirements into your personal remote task list. The `bsub` command automatically uses the resource requirements of the job from the remote task lists. See *'Managing Your Task List' on page 53* for more information about displaying task lists and putting tasks into your remote task list.

Resource Reservation

When a job is dispatched, the system assumes that the resources that the job consumes will be reflected in the load information. However, many jobs often do not consume the resources they require when they first start. Instead, they will typically use the resources over a period of time. For example, a job requiring 100MB of swap space is dispatched to a host having 150MB of available swap space. The job starts off initially allocating 5MB, gradually increasing the amount consumed to 100MB over a 30-minute period. During this period, another job requiring more than 50MB of swap space should not be started on the same host to avoid overcommitting the resource.

When submitting a job, you can specify the amount of resources to be reserved through the resource usage section of resource requirement string argument to the `bsub`

6 Submitting Batch Jobs

command. The syntax of the resource reservation in the `rusage` section of resource requirement string is:

```
res=value[:res=value]...[:res=value][:duration=value][:decay=valu  
e]
```

The *res* parameter can be any load index. The *value* parameter is the initial reserved amount. If *res* or *value* is not given, the default is to not reserve that resource.

The *duration* parameter is the time period within which the specified resources should be reserved. It is specified in minutes by default. If the value is followed by the letter 'h', it is specified in hours. For example, '`duration=30`' and '`duration=2h`' specify a duration of 30 minutes and two hours respectively. If *duration* is not specified, the default is to reserve the total amount for the lifetime of the job.

The *decay* parameter indicates how the reserved amount should decrease over the *duration*. A value of 1, '`decay=1`', indicates that system should linearly decrease the amount reserved over the duration. The default *decay* value is 0, which causes the total amount to be reserved for the entire *duration*. Values other than 0 or 1 are unsupported. If *duration* is not specified *decay* is ignored.

When deciding whether to schedule a job on a host, the LSF Batch system considers the reserved resources of jobs that have previously started on that host. For each load index, the amount reserved by all jobs on that host is summed up and subtracted (or added if the index is *increasing*) from the current value of the resources as reported by the LIM to get amount available for scheduling new jobs:

```
available amount = current value - reserved amount for all jobs
```

For example:

```
% bsub -R "rusage[tmp=30:duration=30:decay=1]" myjob
```

will reserve 30MB of `/tmp` space for the job. As the job runs, the amount reserved will decrease at approximately 1 megabyte/minute such that the reserved amount is 0 after 30 minutes.

The queue level resource requirement parameter `RES_REQ` may also specify the resource reservation. If a queue reserves certain amount of a resource, you cannot use

the `-R` option of the `bsub` command to reserve a greater amount of that resource. For example, if the output of `bqueues -l` command contains:

```
RES_REQ: rusage[mem=40:swp=80:tmp=100]
```

the following submission will be rejected since the requested amount of certain resource(s) exceeds queue's specification:

```
% bsub -R "rusage[mem=50:swp=100]" myjob
```

The amount of resources reserved on each host can be viewed through the `-l` option of the `bhosts` command.

Host Selection

If you want to restrict the set of candidate hosts for running your batch job, use the `-m` option to `bsub`.

```
% bsub -q idle -m "hostA hostD hostB" myjob
```

This command submits `myjob` to the `idle` queue and tells LSF Batch to choose one host from `hostA`, `hostD` and `hostB` to run the job. All other LSF Batch scheduling conditions still apply, so the selected host must be eligible to run the job.

If you have applications that need specific resources, it is more flexible to create a new Boolean resource and configure that resource for the appropriate hosts in the LSF cluster. This must be done by the LSF administrator. If you specify a host list using the `-m` option to `bsub`, you must change the host list every time you add a new host that supports the desired resources. By using a Boolean resource, the LSF administrator can add, move or remove resources without forcing users to learn about changes to resource configuration.

Host Preference

When several hosts can satisfy the resource requirements of a job, the hosts are ordered by load. However, in certain situations it may be desirable to override this behaviour to give preference to specific hosts, even if they are more heavily loaded.

For example, you may have licensed software which runs on different groups of hosts, but prefer to run on a particular host group because the jobs will finish faster, thereby freeing the software license to be used by other jobs.

Another situation arises in clusters consisting of dedicated batch servers and desktop machines which can also run jobs when no user is logged in. You may prefer to run on the batch servers and only use the desktop machines if no server is available.

The `-m` option of the `bsub` command allows you to specify preference by using '+' after the hostname. The special hostname, `others`, can be used to refer to all the hosts that are not explicitly listed. For example:

```
% bsub -R "solaris && mem> 10" -m "hostD+ others" myjob
```

will select all `solaris` hosts having more than 10 megabytes of memory available. If host `hostD` satisfies this criteria, it will be picked over any other host which otherwise meets the same criteria. If `hostD` does not satisfy the criteria, the least loaded host among the others will be selected. All the other hosts are considered as a group and are ordered by load.

You can specify different levels of preference by specifying a number after the '+'. The larger the number, the higher the preference for that host or host group. For example:

```
% bsub -m "groupA+2 groupB+1 groupC" myjob
```

gives first preference to hosts in `groupA`, second preference to hosts in `groupB` and last preference to those in `groupC`. The ordering within a group is still determined by the load. You can use the `bmgroup` command to display the host groups configured in the system.

Note

A queue may also define the host preference for jobs via `HOSTS` parameter. The queue specification is ignored if a job specifies its own preference.

You can also exclude a host by specifying a resource requirement using `hname` resource:

```
% bsub -R "hname!=hostb && type==sgi6" myjob
```

Resource Limits

Resource limits are constraints you or your LSF administrator can specify to limit the use of resources. Jobs that consume more than the specified amount of a resource are signalled or have their priority lowered.

Resource limits can be specified either at the queue level by your LSF administrator or at the job level when you submit a job. Resource limits specified at the queue level are hard limits while those specified with job submission are soft limits. See `setrlimit(2)` man page for concepts of hard and soft limits.

The following resource limits can be specified to the `bsub` command:

-c `cpu_limit[/host_spec]`

Set the soft CPU time limit to `cpu_limit` for this batch job. The default is no limit. This option is useful for preventing erroneous jobs from running away, or to avoid using up too many resources. A `SIGXCPU` signal is sent to all processes belonging to the job when it has accumulated the specified amount of CPU time. If the job has no signal handler for `SIGXCPU`, this causes it to be killed. LSF Batch keeps track of the CPU time used by all processes of the job.

`cpu_limit` is in the form `[hour:]minute`, where `minute` can be greater than 59. So, 3.5 hours can either be specified as `3:30` or `210`. The CPU limit is scaled by the host CPU factors of the submitting and execution hosts. This is done so that the job does approximately the same amount of processing for a given CPU limit, even if it is sent to a host with a faster or slower CPU. For example, if a job is submitted from a host with a CPU factor of 2 and executed on a host with a CPU factor of 3, the CPU time limit is multiplied by $2/3$ because the execution host can do the same amount of work as the submission host in $2/3$ of the time.

6 Submitting Batch Jobs

The optional `host_spec` specifies a host name or a CPU model name defined by LSF. The `lsinfo` command displays CPU model information. If `host_spec` is not given, the CPU limit is scaled based on the `DEFAULT_HOST_SPEC` shown by the `bparams -l` command. (If `DEFAULT_HOST_SPEC` is not defined, the fastest batch host in the cluster is used as the default.) If `host_spec` is given, the appropriate CPU scaling factor for the specified host or CPU model is used to adjust the actual CPU time limit at the execution host. The following example specifies that `myjob` can run for 10 minutes on a DEC3000 host, or the corresponding time on any other host:

```
% bsub -c 10/DEC3000 myjob
```

-W run_limit[/host_spec]

Set the wall-clock run time limit of this batch job. The default is no limit. If the accumulated time the job has spent in the RUN state exceeds this limit, the job is sent a USR2 signal. If the job does not terminate within 10 minutes after being sent this signal, it is killed. `run_limit` and `host_spec` have the same format as the argument to the `bsub -c` option.

-F file_limit

Set a per-process (soft) file size limit for each process that belongs to this batch job. If a process of this job attempts to write to a file such that the file size would increase beyond `file_limit` kilobytes, the kernel sends that process a SIGXFSZ signal. This condition normally terminates the process, but may be caught. The default is no soft limit.

-D data_limit

Set a per-process (soft) data segment size limit for each process that belongs to this batch job. An `sbrk()` or `malloc()` call to extend the data segment beyond `data_limit` kilobytes returns an error. The default is no soft limit.

-S stack_limit

Set a per-process (soft) stack segment size limit for each process that belongs to this batch job. An `sbrk()` call to extend the stack segment beyond `stack_limit` kilobytes causes the process to be terminated. The default is no soft limit.

-C core_limit

Set a per-process (soft) core file size limit for each process that belongs to this batch job. On some systems, no core file is produced if the image for the

process is larger than `core_limit` kilobytes. On other systems only the first `core_limit` kilobytes of the image are dumped. The default is no soft limit.

-M `mem_limit`

Set the per-process (soft) process resident set size limit to `mem_limit` kilobytes for all processes that belong to this batch job. Exceeding this limit when free physical memory is in short supply results in a low scheduling priority being assigned to the process. That is, the process is reniced. The default is no soft limit. On HP-UX and Sun Solaris 2.x, a resident set size limit cannot be set, so this option has no effect.

Pre-Execution Commands

Some batch jobs require resources that LSF does not directly support. For example, a batch job may need to reserve a tape drive or check for the availability of a software license.

The `-E pre_exec_command` option to the `bsub` command specifies an arbitrary command to run before starting the batch job. When LSF Batch finds a suitable host on which to run a job, the pre-execution command is executed on that host. If the pre-execution command runs successfully, the batch job is started.

An alternative to using the `-E pre_exec_command` option is for the LSF administrator to set up a queue level pre-execution command. See '*Queue-Level Pre-/Post-Execution Commands*' on page 224 of the *LSF Batch Administrator's Guide* for more information.

By default, the pre-execution command is run under the same user ID, environment, and home and working directories as the batch job. For queue-level pre-execution commands, you can specify a different user ID by defining the `LSB_PRE_POST_EXEC_USER` variable. If the pre-execution command is not in your normal execution path, the full path name of the command must be specified.

For parallel batch jobs, the pre-execution command is run on the first selected host.

The pre-execution command returns information to LSF Batch using the exit status. If the pre-execution command exits with non-zero status, the batch job is not dispatched.

6 Submitting Batch Jobs

The job goes back to the `PEND` state, and LSF Batch tries to dispatch another job to that host. The next time LSF Batch tries to dispatch jobs this process is repeated.

LSF Batch assumes that the pre-execution command runs without side effects. For example, if the pre-execution command reserves a software license or other resource, you must take care not to reserve the same resource more than once for the same batch job.

The following example shows a batch job that requires a tape drive. The `tapeCheck` program is a site specific program that exits with status zero if the specified tape drive is ready, and one otherwise:

```
% bsub -E "/usr/local/bin/tapeCheck /dev/rmt01" myjob
```

Job Dependencies

Some batch jobs depend on the results of other jobs. For example, a series of jobs could process input data, run a simulation, generate images based on the simulation output, and finally, record the images on a high-resolution film output device. Each step can only be performed when the previous step completes and all subsequent steps must be aborted if any step fails.

The `-w depend_cond` option to the `bsub` command specifies a dependency condition, which is a logical expression based on the execution states of preceding batch jobs. When the `depend_cond` expression evaluates to `TRUE`, the batch job can be started. Complex conditions can be written using the logical operators `'&&'` (AND), `'|'` (OR), `'!'` (NOT) and parentheses `'()'`.

If any one of the depended batch jobs is not found, `bsub` fails and the job is not submitted.

Inter-job dependency scheduling can be based on specific job exit status, so that a suitable recovery job can be initiated in case of specific types of job failures. The exit condition in the dependency string (specified in the `-w` option of `bsub`) can be triggered on particular exit code(s) of the dependant job. Relational operators can be used when a job needs to be triggered on a range of exit codes.

If there is a space character, a logic operator or parentheses in the expression string, the string must be enclosed in single or double quotes (' or ") to prevent the shell from interpreting the special characters.

Batch jobs are identified by job ID number or job name. The job ID number is displayed by the `bsub` command when the job is submitted. The job name is a string specified by the `-J job_name` option.

In job dependency expressions, numeric job names must be enclosed in quotes.

UNIX

Note that a numeric job name should be doubly quoted, e.g. `-w " ' 210 ' "`, since the UNIX shell treats `-w " 210 "` the same as `-w 210`.

Job names refer to jobs submitted by the same user. If more than one of your jobs has the same name, the condition is tested on the last job submitted with that name.

A wildcard character `*` can be specified at the end of a job name to indicate all jobs matching the name. For example, `jobA*` will match `jobA`, `jobA1`, `jobA_test`, `jobA.log` etc. There must be at least one match.

The conditions that can be tested are:

started({jobID | jobName})

If the specified batch job has started running or has run to completion, the condition is TRUE; that is, the job is not in the `PEND` or `PSUSP` state, and also is not currently running the pre-execution command if the `bsub -E` option was specified.

done({jobID | jobName})

If the specified batch job has completed successfully and is in the `DONE` state, the condition is TRUE. Otherwise, it is FALSE.

exit({jobID | jobName})

If the specified batch job has terminated abnormally and is in the `EXIT` state, the condition is TRUE. Otherwise, it is FALSE.

exit({jobID | jobName}, [op] code)

If the specified job has terminated with the exit code specified by *code*, or with an exit code satisfying the relationship expressed by *op code*, the condition is

6 Submitting Batch Jobs

TRUE. Otherwise, it is FALSE. When a batch job is killed while pending, it is assigned a special exit code of 512.

The *op* variable may be any of the relational operators '>', '>=', '<', '<=', '==', '!='. The *code* variable is numeric, representing a job exit code.

`ended({jobID | jobName})`

If the specified batch job has finished (either in the EXIT or DONE state), the condition is TRUE. Otherwise, it is FALSE.

`{jobID | jobName}`

Specifying only `jobID` or `jobName` is equivalent to `done({jobID | jobName})`. If the specified batch job has completed successfully and is in the DONE state, the condition is TRUE. Otherwise, it is FALSE.

Job Dependency Examples

```
done(312) && (started(Job2) || exit(Job3))
```

The submitted job will not start until job 312 has completed successfully, and either the job named `Job2` has started or the job named `Job3` has terminated abnormally.

```
1532 || jobName2 || ended(jobName3*)
```

The submitted job will not start until either job 1532 has completed, the job named `jobName2` has completed, or all jobs with names beginning with `jobName3` have finished.

```
exit (34334, 12)
```

The submitted job will not start until job 34334 finishes with an exit code of 12.

```
exit (myjob, < 30)
```

The submitted job will not start until `myjob` finishes with an exit code lower than 30.

Note

If you require more extensive dependencies, for example, calendar or event dependencies, you may want to examine the LSF JobScheduler product of LSF Suite.

Remote File Access

LSF is usually used in networks with shared file space. When shared file space is not available, LSF can copy needed files to the execution host before running the job, and copy result files back to the submission host after the job completes.

The `-f "[lfile op [rfile]]"` option to the `bsub` command copies a file between the submission host and the execution host. `lfile` is the file name on the submission host, and `rfile` is the name on the execution host. `op` is the operation to perform on the file. `lfile` and `rfile` can be absolute or relative file path names. If one of the files is not specified, it defaults to the other, which must be given.

The `-f` option may be repeated to specify multiple files.

`op` must be surrounded by white space. The possible values for `op` are:

- > `lfile` on the submission host is copied to `rfile` on the execution host before job execution. `rfile` is overwritten if it exists
- < `rfile` on the execution host is copied to `lfile` on the submission host after the job completes. `lfile` is overwritten if it exists
- << `rfile` is appended to `lfile` after the job completes. `lfile` is created if it does not exist
- ><, <> equivalent to performing the > and then the < operation. `lfile` is copied to `rfile` before the job executes, and `rfile` is copied back (replacing the previous `lfile`) after the job completes. '<>' is the same as '><'

You must include `lfile` with `op`, otherwise it will result in a syntax error. When `rfile` is not given, it is assumed to be the same as `lfile`.

If the input file specified with the `-i` argument to `bsub` is not found on the execution host, the file is copied from the submission host using LSF's remote file access facility and is removed from the execution host after the job finishes.

6 Submitting Batch Jobs

The output files specified with the `-o` and `-e` arguments to `bsub` are created on the execution host, and are not copied back to the submission host by default. You can use the remote file access facility to copy these files back to the submission host if they are not on a shared file system. For example, the following command stores the job output in the `job_out` file and copies the file back to the submission host:

```
% bsub -o job_out -f "job_out <" myjob
```

If the submission and execution hosts have different directory structures, you must ensure that the directory where *rfile* and *lfile* will be placed exists. LSF tries to change the directory to the same path name as the directory where the `bsub` command was run. If this directory does not exist, the job is run in your home directory on the execution host.

You should specify *rfile* as a file name with no path when running in non-shared file systems; this places the file in the job's current working directory on the execution host. This way the job will work correctly even if the directory where the `bsub` command is run does not exist on the execution host. Be careful not to overwrite an existing file in your home directory.

For example, to submit `myjob` to LSF Batch, with input taken from the file `/data/data3` and the output copied back to `/data/out3`, run the command:

```
% bsub -f "/data/data3 > data3" -f "/data/out3 < out3" myjob data3 out3
```

To run the job `batch_update`, which updates the `batch_data` file in place, you need to copy the file to the execution host before the job runs and copy it back after the job completes:

```
% bsub -f "batch_data <>" batch_update batch_data
```

LSF Batch uses the `lsrnp(1)` command to transfer files. `lsrnp` contacts the RES on the remote host to perform the file transfer. If the RES is not available, `rcp(1)` is used. Because LSF client hosts do not run the RES daemon, jobs that are submitted from client hosts should only specify the `-f` option to `bsub` if `rcp` is allowed. You must set up the permissions for `rcp` if account mapping is used.

Start and Termination Time

If you do not want LSF Batch to start your job immediately, use the `bsub -b` option to specify the time after which the job should be dispatched.

```
% bsub -b 5:00 myjob
```

The submitted job remains pending until after the local time on the LSF master host reaches 5 A.M. You can also specify a time after which the job should be terminated with the `-t` option to `bsub`. The command

```
% bsub -b 11:12:5:40 -t 11:12:20:30 myjob
```

submits `myjob` to the default queue to start after November 12 at 05:40 A.M. If the job is still running on Nov 12 at 8:30 P.M., it is killed.

Parallel Jobs

LSF Batch can allocate more than one host or processor to run a job and automatically keeps track of the job status, while a parallel job is running. To submit a parallel job, use the `-n` option of `bsub`:

```
% bsub -n 10 myjob
```

This command submits `myjob` as a parallel job. The job is started when 10 job slots are available.

For parallel jobs, LSF Batch only starts one controlling process for the batch job. This process is started on the first host in the list of selected hosts. The controlling process is responsible for starting the actual parallel components on all the hosts selected by LSF Batch.

LSF Batch sets a number of environment variables for each batch job. The variable `LSB_JOBID` is set to the LSF Batch job ID number as printed by `bsub`. The `LSB_HOSTS` variable is set to the names of the hosts running the batch job. For a sequential job, `LSB_HOSTS` is set to a single host name. For a parallel batch job, `LSB_HOSTS` contains

6 Submitting Batch Jobs

the complete list of hosts that LSF Batch has allocated to that job. Parallel batch jobs must get the list of hosts from the `LSB_HOSTS` variable and start up all of the job components on the allocated hosts.

In the `myjob` example above, LSF Batch starts `myjob` on the first host. `myjob` reads the `LSB_HOSTS` environment variable to get the list of hosts and uses the `RES` to execute subtasks on those hosts.

LSF includes scripts for running PVM, P4, and MPI parallel programs as batch jobs. See *'Parallel Jobs' on page 181* and the `pvmjob(1)`, `p4job(1)`, and `mpijob(1)` manual pages for more information.

The following features support parallel jobs running through the LSF Batch system.

Minimum and Maximum Number of Processors

When submitting a parallel job that requires multiple processors, you can specify the minimum number and maximum number of the processors using `-n` option to the `bsub` command. The syntax of the `-n` option is:

```
bsub -n min_proc[,max_proc] <other bsub options>
```

If `max_proc` is not specified then it is assumed to be equal to `min_proc`. For example:

```
% bsub -n 4,16 myjob
```

At most, 16 processors can be allocated to this job. If there are less than 16 processors eligible to run the job, this job can still be started as long as the number of eligible processors is greater than 4. Once the job gets started, no more processors will be allocated to it even though more may be available later on.

If the specified maximum number is greater than the value of `PROCLIMIT` defined for the queue to which the job is submitted, the job will be rejected.

Specifying Locality

Sometimes you need to control how the selected processors for a parallel job are distributed across the hosts in the cluster. You are able to specify "select all the

processors for this parallel batch job on the same host", or "do not chose more than n processor on one host" by using the 'span' section in the -R option string. For example:

```
% bsub -n 4 -R "span[hosts=1]" my_job
```

This job should be dispatched to a multiprocessor that has at least 4 processors currently eligible to run the 4 components of this job.

```
% bsub -n 4 -R "span[ptile=1]" myjob
```

This job should be dispatched to 4 hosts even though some of the 4 hosts may have more than one processor currently available.

Note

The queue may also define the locality for parallel jobs using RES_REQ parameter. The queue specification is ignored if your job specifies its own locality.

A parallel job may span multiple hosts, with a specifiable number of processes allocated to each host. Thus, a job may be scheduled onto a single multiprocessor host to take advantage of its efficient shared memory, or spread out onto multiple hosts to take advantage of their aggregate memory and swap space. Flexible spanning may also be used to achieve parallel I/O.

The span section of the resource requirement string can specify a processor tiling factor, ptile:

```
span[ptile=value]
```

The value is a number greater than 0 indicating that up to <value> processor(s) on each host should be allocated to the job regardless of how many processors the host possesses. For example:

```
% bsub -n 4 -R "span[ptile=2]" myjob
```

This job should be dispatched to 2 hosts with 2 processors on each host allocated for the job. Each host may have more than 2 processors available.

```
% bsub -n 4 -R "span[ptile=3]" myjob
```

6 Submitting Batch Jobs

In this case, the job must be dispatched to 2 hosts. It takes 3 processors on the first host and 1 processor on the second host.

Job Arrays

The LSF Batch system provides a structure called a job array, which allows multiple jobs to be created with a single job submission (`bsub`). A job array is a series of independent batch jobs, all of which share the same job ID and submission parameters (resource requirements). The job array elements are referenced using an array index. The dimension and structure of the job array are defined as part of the job name when the job is submitted.

Job array elements (jobs) are scheduled to run independently of each other, using the various policies that govern a user's jobs within the LSF system. After a job array is submitted, the resource requirements for individual jobs and for the entire array are modified using the `bmod` command (see '*Job Array Modification*' on page 133). Individual jobs and the entire array are controlled using the `bstop`, `bresume`, and `bkill` commands (see '*Controlling Job Arrays*' on page 129). The status and history of a job array and its jobs are viewed using the `bjobs` and `bhist` commands (see '*Tracking Job Arrays*' on page 126).

The default maximum size of a job array is 1000 jobs, but this can be increased to 2046 jobs. The `MAX_JOB_ARRAY_SIZE` parameter specified in the `lsb.params` file sets the maximum size of a job array.

This section discusses the following topics:

- Creating a Job Array
- Array Job Dependencies
- Handling Input/Output/Error Files for Job Arrays

Additional topics about job arrays:

- '*Tracking Job Arrays*' on page 126
- '*Controlling Job Arrays*' on page 129

- 'Job Array Modification' on page 133

Creating a Job Array

A job array is created at the time of submission. The job name field of the `bsub` command is extended to specify the elements of the array. Each element of the array corresponds to a single job and is identified by an index which must be a positive integer.

The index values in an array do not have to be consecutive, and a combination of individual index values and index ranges are used to define a job array. The following command creates a job array with the name `myJobArray` consisting of 100 elements with indices 1 through 100.

```
% bsub -J "myJobArray[1, 2, 3, 4-50, 51, 52, 53-100]"
```

The array elements (jobs) are named `myJobArray[1]`, `myJobArray[2]`, ..., `myJobArray[n]`, ..., `myJobArray[100]`.

Syntax

```
% bsub -J "jobArrayName[indexList, . . .]" command
```

Note

One blank space must be entered between the `-J` switch and the first quote in the job array specification

The job array specification must be enclosed in double quotes

The square brackets, [], around the `indexList` must be entered exactly as shown.

Note: The job array syntax breaks the convention of using square brackets to indicate optional items.

`jobArrayName`

Specifies a user defined string used to name the job array. Any combination of the following characters make up a valid `jobArrayName`:

`a-z` | `A-Z` | `0-9` | `.` | `-` | `_`

6 Submitting Batch Jobs

`indexList`

Specifies the dimension, structure, and indices of the job array in the following format:

```
indexList = start [- end [: step]]
```

`start`

A unique positive integer specifying the start of a range of job array indices. If a start value is specified without an end value, start specifies an individual index in the job array.

`end`

A unique positive integer specifying the end of a range of job array indices.

`step`

A positive integer specifying the value to increment the index values for the preceding range. If omitted, the default value is 1.

Examples of `indexList` specifications

- `[1]` specifies 1 job with the index of 1. Since no end value is specified, the start value is the index.
- `[1, 2, 3, 4, 5]` specifies 5 jobs with indices 1 through 5.
- `[1-10]` specifies 10 jobs with indices 1 through 10. Since no step value is specified, the default step is 1. The index values are determined by starting at 1 and adding 1, the default step value, to the current index. The index values are not incremented past the end value.
- `[10-20:2]` specifies 6 jobs with indices 10, 12, 14, 16, 18, and 20. The step value is 2. Index values are determined by starting at 10 and adding 2 to the current index. The index values are not incremented past the end value.
- `[1, 2, 3, 4, 5, 6-10, 27, 100-200:50, 201, 202]` specifies 16 jobs with indices of 1-10, 27, 100, 150, 200, 201, and 202.

LSB_JOBINDEX Environment Variable

The environment variable `LSB_JOBINDEX` is set when each job array element is dispatched. Its value corresponds to the job array index. Typically this variable is used within a script to select the job command to be performed based on the job array index. For example:

```
if [$LSB_JOBINDEX -eq 1]; then
cmd1
fi
if [$LSB_JOBINDEX -eq 2]; then
cmd2
fi
```

Array Job Dependencies

Since each job array has the same set of submission parameters, it is not possible to set up job dependencies between elements of the same array. Similar behaviour can be achieved by creating two job arrays and using array job dependencies. For example, suppose you want to have an array with 100 elements, where the first 50 elements must be run before next 50. This can be achieved with the following submissions:

```
% bsub -J "myJob[1-50]" cmd
Job <101> submitted to default queue <normal>.

% bsub -w "done(101)" -J "myJob[51-100]" cmd
Job <102> submitted to queue <normal>.
```

The second job array, 102, will wait for the successful completion of all jobs in the array due the `done(101)` dependency. Note that two job arrays can have the same name, but have a different number of elements in each. Each job array is handled independently of the other.

A job or job array can also depend on the partial completion of another array. One of the dependency condition functions listed in the Table can be used to evaluate the number of jobs of a job array in a given job state. The “op” in *Table 6* is one of the strings

6 Submitting Batch Jobs

"==", ">", "<", ">=", or "<=". "num" is a non-negative integer. A special string "*" can be used in place of "num" to mean "all"..

Table 6. Dependency Condition Functions

Function	Description
numrun(array_jobId, op num)	TRUE if RUN counter satisfies test
numpend(array_jobId, op num)	TRUE if PEND counter satisfies test
numdone(array_jobId, op num)	TRUE if DONE counter satisfies test
numexit(array_jobId, op num)	TRUE if EXIT counter satisfies test
numended(array_jobId, op num)	TRUE if DONE+EXIT counter satisfies test
numhold(array_jobId, op num)	TRUE if PSUSP counter satisfies test
numstart(array_jobId, op num)	TRUE if RUN+SSUSP+USUSP counters satisfies test

In the following example, the elements in job array 202 will be scheduled when 10 or more elements in job array 201 have completed successfully.

```
% bsub -J "myJob[1-50]" cmd
```

```
Job <201> submitted to default queue <normal>.
```

```
% bsub -w "numdone(201,>=10)" -J "myJob[51-100]" cmd
```

```
Job <202> submitted to default queue <normal>.
```

Handling Input/Output/Error Files for Job Arrays

If input, output, or error files are specified for the array, then all elements will share these files. In order to separate the I/O of each element, special strings can be inserted in the I/O file specification to indicate the job ID or the array index of the element. The

string "%J" and "%I" are expanded at job execution time into the job ID and array index (respectively), when found in the input, output or error specification. Both "%I" and "%J" may be specified simultaneously. For example:

```
% bsub -J "render[1-5]" -i "frame.%I" renderit
Job <200> submitted to default queue <normal>.
```

would result in an array with 5 elements: `render[1]`, .. `render[5]`, whose input files are "frame.1", "frame.2", ..., "frame.5", respectively.

Specifying a Share Account

If the cluster uses fairshare to determine the rate of resource allocation, then the order in which jobs are dispatched will be different from the default first-come-first-served (FCFS) policy. A user cannot control the priority assigned by the system when fairshare is enabled. However, if a user belongs to more than one group in the share tree and hence has multiple share accounts, then the job can be associated with a particular share account. This can be done by using the `-G` option of `bsub(1)`. For example, if a user is member of both the 'test' and 'development' groups, a job can be submitted which uses the users share account in the test group by:

```
% bsub -G test myJob
```

Note that the user must have an account under the group specified, otherwise the job is rejected. Use `bugroup -l` to find out if you belong to a group.

UNIX

Re-initializing Job Environment on the Execution Host

By default LSF Batch copies the environment of the job from the submission host when the job is submitted. The environment is recreated on the execution host when the job is started. This is convenient, in many cases, because the job runs as if it were run interactively on the submission host.

6 Submitting Batch Jobs

There are cases where you want to use a platform specific or host specific environment to run the job, rather than using the same environment as on the submission host. For example, you may want to set up different search paths on the execution host.

The `-L shell` option to the `bsub` command causes LSF Batch to emulate a login on the execution host before starting your job. This makes sure that the login start up files (`.profile` for `/bin/sh`, or `.cshrc` and `.login` for `/bin/csh`) are sourced before the job is started. The `shell` argument specifies the login shell to use.

```
% bsub -L /a/b/shell myjob
```

Job <1234> is submitted to default queue <normal>.

This tells LSF Batch to use `/a/b/shell` as the login shell to reinitialize the environment.

Note

This does not affect the shell under which the job is run. When a login shell is specified with the `-L shell` option to the `bsub` command, that shell is only used as a login shell to set the environment. The job is run using `/bin/sh`, unless you specify otherwise as described in 'Running a Job Under a Particular Shell' on page 116. For example, if your job script is written in `/bin/sh` and your regular login shell is `/bin/csh`, you can run your job under `/bin/sh` but use `/bin/csh` to reinitialize the job environment by sourcing your `.cshrc` and `.login` files.

Other `bsub` Options

This section lists some other `bsub` options. For details on these options see the `bsub(1)` manual page.

-x

The job must run exclusively on a host. The job is started on a host that has no other LSF Batch jobs running on it. The host is locked (status `lockU`) while this job is running so that no other LSF jobs are sent to the host.

-
- r
Specify that the job is rerunnable. See '*Automatically Rerunning and Restarting Jobs*' on page 174.
 - B
Send email to the job submitter when the job begins executing.
 - H
The job is submitted so that it is not scheduled until it is explicitly released by the user or administrator. The job immediately goes into the PSUSP state instead of the PEND state. A `brresume(1)` command would cause the job to go into the PEND state, where it could be scheduled.

This feature is useful when a job must wait on a condition which cannot be detected through LSF. The user or administrator can manually resume the job when the condition is satisfied, allowing it to be scheduled.
 - I
An interactive batch job is submitted to the LSF Batch system. See '*Interactive Batch Job Support*' on page 145 for more details.
 - k "`checkdir[interval]`"
Specify the checkpoint directory and interval. See '*Submitting Checkpointable Jobs*' on page 169.
 - P `project`
Associate a project name with a job. Project names are logged in the `lsb.acct` file. You can use the `bacct` command to gather accounting information on a per-project basis.

On systems running IRIX 6, before the submitted job begins execution, a new array session is created and the project Id corresponding to the project name is assigned to the session.
 - K
Force the synchronous execution of a job: the `bsub` command will not return until the specified job finishes running.

This is useful in cases where the completion of the job is required in order to proceed, such as a job script. If the job needs to be rerun due to transient failures, the command will return after the job finishes successfully.

6 Submitting Batch Jobs

For example:

```
% ./bsub -K myJob
Job <205> is submitted to default queue < normal>.
<< Waiting for dispatch ...>>
```

This will cause the `bsub` command to wait until the job is completed before returning. `bsub` will exit with the same exit code as the application, so that job submission scripts can take appropriate actions based on any failure conditions.

Job Scripts

You can build a job file one line at a time, or create it from another file, by running `bsub` without a command to submit. When you do this, you start an interactive session where `bsub` reads command lines from the standard input and submits them as a single batch job. You are prompted with `bsub>` for each line.

Examples

```
UNIX % bsub -q simulation
bsub> cd /work/data/myhomedir
bsub> myjob arg1 arg2 .....
bsub> rm myjob.log
bsub> ^D
Job <1234> submitted to queue <simulation>.
```

In this case, the three command lines are submitted to LSF Batch and run as a Bourne shell (`/bin/sh`) script. Note that only valid Bourne shell command lines are acceptable in this case. Here is another example:

```
% bsub -q simulation < command_file
Job <1234> submitted to queue <simulation>.
```

`command_file` must contain Bourne shell command lines.

NT

```
C:\> bsub -q simulation
bsub> cd \\server\data\myhomedir
bsub> myjob arg1 arg2 .....
bsub> del myjob.log
bsub> ^Z
Job <1234> submitted to queue <simulation>.
```

In this case, the three command lines are submitted to LSF Batch and run as a batch file (.BAT). Note that only valid Windows batch file command lines are acceptable in this case. Here is another example:

```
% bsub -q simulation < command_file
Job <1234> submitted to queue <simulation>.
```

`command_file` must contain Windows batch file command lines.

Embedded Submission Options

You can specify job submission options in the script read from the standard input by the `bsub` command using lines starting with '#BSUB':

```
% bsub -q simulation
bsub> #BSUB -q test
bsub> #BSUB -o outfile -R "mem>10"
bsub> myjob arg1 arg2
bsub> #BSUB -J simjob
bsub> ^D
Job <1234> submitted to queue <simulation>.
```

There are a few things to note:

- Command line options override embedded options, therefore the job is submitted to the simulation queue rather than the test queue
- Submission options can be specified anywhere in the standard input. In the above example, the `-J` option to `bsub` is specified after the command to be run
- More than one option can be specified on one line, as shown in the above example

6 Submitting Batch Jobs

As a second example, you can redirect a script to the standard input of the `bsub` command:

```
% bsub < myscript
Job <1234> submitted to queue <test>.
```

The `myscript` file contains job submission options as well as command lines to execute. When the `bsub` command reads a script from its standard input, the script file is actually spooled by the LSF Batch system; therefore, the script can be modified right after `bsub` returns for the next job submission.

When the script is specified on the `bsub` command line, the script is not spooled:

```
% bsub myscript
Job <1234> submitted to default queue <normal>.
```

In this case the command line `myscript` is spooled by LSF Batch, instead of the contents of the `myscript` file. Later modifications to the `myscript` file can affect the job's behaviour.

UNIX

Running a Job Under a Particular Shell

By default, LSF runs batch jobs using the Bourne (`/bin/sh`) shell. You can specify the shell under which the job is run. This is done by specifying an interpreter in the first line of the script.

```
% bsub
bsub> #!/bin/csh -f
bsub> set coredump='ls |grep core'
bsub> if ( "$coredump" != "" ) then
bsub> mv core core.`date | cut -d" " -f1`
bsub> endif
bsub> myjob
bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

The `bsub` command must read the job script from the standard input to set the execution shell.

If you do not specify a shell in the script, the script is run using `/bin/sh`. If the first line of the script starts with a `#` not immediately followed by a `!`, then `/bin/csh` is used to run the job. For example:

```
% bsub
bsub> # This is a comment line. This tells the system
      to use /bin/csh to
bsub> # interpret the script.
bsub>
bsub> setenv DAY `date | cut -d" " -f1`
bsub> myjob
bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

If running jobs under a particular shell is frequently required, you can specify an alternate shell using a command-level job starter and run your jobs interactively. See *‘Command-Level Job Starters’* on page 144 for detailed information.

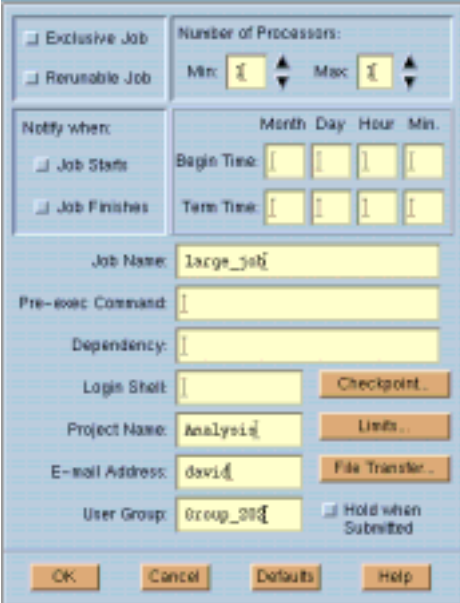
Submitting Jobs Using the Job Submission GUI

LSF Batch provides a GUI for submitting jobs. The main window of `xbsub` was shown in the figure *‘xbsub Job Submission Window’* on page 23. All the job submission options can be selected using the GUI.

6 Submitting Batch Jobs

Detailed parameters can be set by clicking the 'Advanced' button. The resulting window is shown in *Figure 11*.

Figure 11. Advanced Parameters of the Job Submission Window



The image shows a software window titled "Advanced Parameters of the Job Submission Window". It contains various input fields and checkboxes for configuring a batch job. The fields are organized into sections: "Exclusive Job" and "Rerunnable Job" checkboxes at the top left; "Number of Processors" with "Min" and "Max" spinners at the top right; "Notify when:" with "Job Starts" and "Job Finishes" checkboxes in the middle left; "Begin Time" and "Term Time" with month, day, hour, and minute spinners in the middle right; "Job Name:" with a text field containing "large_job"; "Pre-exec Command:" with an empty text field; "Dependency:" with an empty text field; "Login Shell:" with an empty text field and a "Checkpoint..." button; "Project Name:" with a text field containing "Analysis" and a "Limits..." button; "E-mail Address:" with a text field containing "david" and a "File Transfer..." button; "User Group:" with a text field containing "Group_000" and a "Hold when Submitted" checkbox; and a bottom row with "OK", "Cancel", "Defaults", and "Help" buttons.

<input type="checkbox"/> Exclusive Job	Number of Processors:
<input type="checkbox"/> Rerunnable Job	Min: [1] [up] [down] Max: [1] [up] [down]
Notify when:	Month Day Hour Min.
<input type="checkbox"/> Job Starts	Begin Time: [] [] [] []
<input type="checkbox"/> Job Finishes	Term Time: [] [] [] []
Job Name:	large_job
Pre-exec Command:	[]
Dependency:	[]
Login Shell:	[] [Checkpoint...]
Project Name:	Analysis [Limits...]
E-mail Address:	david [File Transfer...]
User Group:	Group_000 <input type="checkbox"/> Hold when Submitted
[OK] [Cancel] [Defaults] [Help]	

7. Tracking Batch Jobs

This chapter describes the commands that report and change the status of your jobs:

- Displaying the current status of batch jobs
- Finding out why jobs are pending or suspended
- Displaying the execution history of jobs
- Checking the output from jobs that have not completed yet
- Killing, suspending and resuming jobs
- Changing the order of your pending jobs within a queue
- Moving jobs to other queues

Displaying Job Status

The `bjobs` command reports the status of LSF Batch jobs.

% `bjobs`

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
3926	user1	RUN	priority	hostf	hostc	verilog	Oct 22 13:51
605	user1	SSUSP	idle	hostq	hostc	Test4	Oct 17 18:07
1480	user1	PEND	priority	hostd		generator	Oct 19 18:13
7678	user1	PEND	priority	hostd		verilog	Oct 28 13:08
7679	user1	PEND	priority	hosta		coreHunter	Oct 28 13:12
7680	user1	PEND	priority	hostb		myjob	Oct 28 13:17

7 Tracking Batch Jobs

The `-a` option displays jobs that completed or exited in the recent past, along with pending and running jobs.

The `-r` option displays only running jobs.

The `-u username` option displays jobs submitted by other users. The special user name `all` displays jobs submitted by all users.

For example, to find out who is running jobs on which hosts enter:

```
% bjobs -u all
```

You can also find jobs on specific queues or hosts, find jobs submitted by specific projects, and check the status of specific jobs using their job IDs or names. See the `bjobs(1)` manual page for more information.

Finding Pending or Suspension Reasons

When you submit a job to LSF Batch, it may be held in the queue before it starts running and it may be suspended while running. The `-p` option to the `bjobs` command displays the reasons a job is pending. Because there can be more than one reason the job is pending or suspended, all reasons that contributed to the pending or suspension are reported. For example:

```
% bjobs -p
7678   user1   PEND   priority   hostD   verilog   Oct 28 13:08
Queue's resource requirements not satisfied:3 hosts;
Unable to reach slave lsbatch server: 1 host;
Not enough job slots: 1 host;
```

The pending reasons will also mention the number of hosts for each condition. To get the specific host names, along with pending reasons, use the `-p` and `-l` options to the `bjobs` command. For example:

% bjobs -lp

Job Id <7678>, User <user1>, Project <default>, Status <PEND>, Queue <priority>, Command <verilog>
 Mon Oct 28 13:08:11: Submitted from host <hostD>, CWD <\$HOME>, Requested Resources <type==any && swp>35>;

PENDING REASONS:

Queue's resource requirements not satisfied: hostb, hostk, hostv;
 Unable to reach slave lsbatch server: hostH;
 Not enough job slots: hostF;

SCHEDULING PARAMETERS:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	0.7	1.0	-	4.0	-	-	-	-	-	-
loadStop	-	1.5	2.5	-	8.0	-	-	-	-	-	-

Note

In a cluster with many hosts (100-200 hosts), it may be too verbose or considered unnecessary to always show the host names with the pending reasons. Therefore, use the bjobs command with the -p option only.

The -l option to the bjobs command displays detailed information about job status and parameters, such as the job's current working directory, parameters specified when the job was submitted, and the time when the job started running.

% bjobs -l 7678

Job Id <7678>, User <user1>, Project <default>, Status <PEND>, Queue <priority>, Command <verilog>
 Mon Oct 28 13:08:11: Submitted from host <hostD>, CWD <\$HOME>, Requested Resources <type==any && swp>35>;

PENDING REASONS:

Queue's resource requirements not satisfied: 3 hosts;
 Unable to reach slave lsbatch server: 1 host;
 Not enough job slots: 1 host;

SCHEDULING PARAMETERS:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	0.7	1.0	-	4.0	-	-	-	-	-	-
loadStop	-	1.5	2.5	-	8.0	-	-	-	-	-	-

7 Tracking Batch Jobs

The `loadSched` and `loadStop` thresholds displayed are those that apply to this job. If the job is pending, the thresholds are taken from the queue. If the job has been dispatched, each threshold is the more restrictive of the queue and execution host thresholds for that load index.

Scheduling is also affected by other queue constraints such as `RES_REQ`, `STOP_COND`, `RESUME_COND`, fairshare policy, and others.

The `-s` option displays the reasons a batch job was suspended. Because the load conditions are constantly changing, the reasons for suspension may be out of date. Once the job is suspended it does not resume execution until its scheduling conditions are met.

```
% bjobs -s
JOBID USER  STAT  QUEUE FROM_HOST EXEC_HOST JOB_NAME  SUBMIT_TIME
605   user1 SSUSP idle  hosta   hostc   Test4    Oct 17 18:07
The host load exceeded the following threshold(s):
Paging rate: pg;
Idle time: it;
```

In the example above, the job was suspended because the paging rate and interactive idle time on the execution host went above the suspending threshold. Even though the paging rate may have dropped back below the scheduling threshold, the job may remain suspended because of another threshold. The job does not resume until all load indices are within their scheduling thresholds.

Monitoring Resource Consumption of Jobs

Jobs submitted through the LSF Batch system have the resources they consume monitored while they are running. The `-l` option of the `bjobs` command displays the current resource usage of the job. This job-level information includes:

- Total CPU time consumed by all processes in the job
- Total resident memory usage in kbytes of all currently running processes in a job
- Total virtual memory usage in kbytes of all currently running processes in a job

- Currently active process group ID in a job
- Currently active processes in a job

The job-level resource usage information is updated at a maximum frequency of every `SBD_SLEEP_TIME` seconds (see *‘The lsb.params File’ on page 193 of the LSF Batch Administrator’s Guide* for the value of `SBD_SLEEP_TIME`). The update is done only if the value for the CPU time, resident memory usage, or virtual memory usage has changed by more than 10 percent from the previous update or if a new process or process group has been created.

```
% bjobs -l 1531
```

```
Job Id <1531>, User <user1>, Project <default>, Status <RUN>, Queue
<priority> Command <example 200>
Fri Dec 27 13:04:14 Submitted from host <hostA>, CWD <$HOME>,
SpecifiedHosts <hostD>;
Fri Dec 27 13:04:19: Started on <hostD>, Execution Home </home/user1
>, Execution CWD </home/user1>;
Fri Dec 27 13:05:00: Resource usage collected.
The CPU time used is 2 seconds.
MEM: 147 Kbytes; SWAP: 201 Kbytes PGID: 8920; PIDs: 8920 8921 8922
```

```
SCHEDULING PARAMETERS:
```

	r15s	rlm	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

Displaying Job History

Sometimes you want to know what has happened to your job since it was submitted. The `bhist` command displays a summary of the pending, suspended and running

7 Tracking Batch Jobs

time of batch jobs. The `-l` option of the `bhist` command prints the time information and a complete history of the scheduling events for each job.

```
%bhist -l 1531
```

```
JobId <1531>, User <user1>, Project <default>, Command< example200>
Fri Dec 27 13:04:14: Submitted from host <hostA> to Queue <priority
>, CWD <$HOME>, Specified Hosts <hostD>;
Fri Dec 27 13:04:19: Dispatched to <hostD>;
Fri Dec 27 13:04:19: Starting (Pid 8920);
Fri Dec 27 13:04:20: Running with execution home </home/user1>, Exe
cution CWD </home/user1>, Execution Pid <8920>
;
Fri Dec 27 13:05:49: Suspended by the user or administrator;
Fri Dec 27 13:05:56: Suspended: Waiting for re-scheduling after bei
ng resumed byuser;
Fri Dec 27 13:05:57: Running;
Fri Dec 27 13:07:52: Done successfully. The CPU time used is 28.3 s
conds.
```

```
Summary of time in seconds spent in various states by Sat Dec 27
13:07:52 1997
```

PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
5	0	205	7	1	0	218

The `-J job_name` option of the `bhist` command displays the history of all LSF Batch jobs with the specified job name. Job names are assigned with the `-J job_name` option of the `bsub` command.

LSF keeps job history information after the job exits, so you can look at the history of jobs that completed in the past. The length of the history depends on how often the LSF administrator cleans up the log files.

By default, `bhist` only displays job history from the current event log file. The `-n` option to the `bhist` command allows users to display the history of jobs that completed a long time ago, and are no longer listed in the active event log.

The LSF Batch system periodically backs up and prunes the job history log. The `-n num_logfiles` option tells the `bhist` command to search through the specified number of log files instead of only searching the current log file. Log files are searched in

reverse time order; for example, the command `bhist -n 3` searches the current event log file and then the two most recent backup files.

Viewing Chronological History

By default the `bhist` command displays information from the job event history file, `lsb.events`, on a per job basis. The `-t` option to `bhist(1)` can be used to display the events chronologically, instead of grouping all events for each job. The `-T` option allows for selecting only those events within a given time range.

For example, the following displays all events which occurred between 14:00 and 15:00 hours on a given day:

```
% bhist -t -T 14:00,14:30
Wed Oct 22 14:01:25: Job <1574> done successfully;
Wed Oct 22 14:03:09: Job <1575> submitted from host to Queue ,
                    CWD , User , Project , Command , Requested
                    Resources ;
Wed Oct 22 14:03:18: Job <1575> dispatched to ;
Wed Oct 22 14:03:18: Job <1575> starting (Pid 210);
Wed Oct 22 14:03:18: Job <1575> running with execution home , E
                    xecution CWD , Execution Pid <210>;
Wed Oct 22 14:05:06: Job <1577> submitted from host to Queue,
                    CWD , User , Project , Command , Requested
                    Resources ;
Wed Oct 22 14:05:11: Job <1577> dispatched to ;
Wed Oct 22 14:05:11: Job <1577> starting (Pid 429);
Wed Oct 22 14:05:12: Job <1577> running with execution home, Ex
                    ecution CWD , Execution Pid <429>;
Wed Oct 22 14:08:26: Job <1578> submitted from host to Queue, C
                    WD , User , Project , Command;
Wed Oct 22 14:10:55: Job <1577> done successfully;
Wed Oct 22 14:16:55: Job <1578> exited;
Wed Oct 22 14:17:04: Job <1575> done successfully;
```

Checking Partial Job Output

The output from an LSF Batch job is normally not available until the job is finished. However, LSF Batch provides the `bpeek` command for you to look at the output the job has produced so far. By default, `bpeek` shows the output from the most recently submitted job; you can also select the job by queue or execution host, or specify the job ID or job name on the command line.

```
% bpeek 1234
<< output from stdout >>
Starting phase 1
Phase 1 done
Calculating new parameters
...
```

Only the job owner can use `bpeek` to see job output. The `bpeek` command will not work on a job running under a different user account.

You can use this command to check if your job is behaving as you expected and kill the job if it is running away or producing unusable results. This could save you time.

Tracking Job Arrays

The status individual elements of a job array can be viewed using the `bjobs` command or the `xlsbatch` GUI. The `JOBID` field will be the same for all elements of the array and the `JOBNAME` field will have the index of the element appended to it i.e `jobName[index]`. The following output shows the result of submitting and viewing the job array through `bjobs`.

```
% bsub -J "myArray[1-5]" sleep 10
Job <212> is submitted to default queue.
```

```
% bjobs
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
212	user1	RUN	default	hostA	hostB	myArray[1]	Jul 25 12:45
212	user1	PEND	default	hostA		myArray[2]	Jul 25 12:45
212	user1	PEND	default	hostA		myArray[3]	Jul 25 12:45
212	user1	PEND	default	hostA		myArray[4]	Jul 25 12:45
212	user1	PEND	default	hostA		myArray[5]	Jul 25 12:45

To display summary information about the number of jobs in the different states in the array, use the `-A` option of `bjobs` as follows.

```
% bjobs -A
JOBID  NAME                               OWNER NJOBS PEND RUN DONE EXIT SSUSP USUSP PSUSP
215    testArray[1-100:2] user1 50    40   5   1   0   0   0   4
```

The history of jobs in the array can be viewed using the `bhist` command. When the `jobId` of an array is specified, the history of each element is displayed.

```
% bhist 212
```

The history of a specific element(s) can be displayed by appending an index specification after the job id. For example:

```
% bhist "212[5]"
```

Displaying Queue and Host Status

The `bqueues` and `bhosts` commands display the number of jobs in a queue or dispatched to a host. For more information on these commands see *'Batch Queues' on page 67* and *'Batch Hosts' on page 79*.

Job Controls

After a job is submitted, you can control it by killing it, suspending it, or resuming it.

7 Tracking Batch Jobs

Killing Jobs

The `bkill` command cancels pending batch jobs and sends signals to running jobs. By default, on UNIX, `bkill` sends the `SIGKILL` signal to running jobs. For example, to kill job 3421 enter:

```
% bkill 3421
Job <3421> is being terminated
```

Before `SIGKILL` is sent, `SIGINT` and `SIGTERM` are sent to give the job a chance to catch the signals and clean up. The signals are forwarded from the `mbatchd` to the `sbatchd`. The `sbatchd` waits for the job to exit before reporting the status. Because of these delays, for a short period of time after the `bkill` command has been sent, `bjobs` may still report that the job is running.

On Windows NT, job control messages replace the `SIGINT` and `SIGTERM` signals, and termination is implemented by the `TerminateProcess()` system call.

Suspending and Resuming Jobs

The `bstop` and `bresume` commands allow you to suspend or resume a job.

To suspend job 3421, enter:

```
% bstop 3421
Job <3421> is being stopped
```

UNIX `bstop` sends the `SIGSTOP` signal to sequential jobs and `SIGTSTP` to parallel jobs. `SIGTSTP` is sent to a parallel job so the master process can trap the signal and pass it to all the slave processes running on other hosts.

NT `bstop` causes the job to be suspended.

To resume the same job, enter:

```
% bresume 3421
Job <3421> is being resumed
```


Suspending a job causes your job to go into `USUSP` state if the job is already started, or to go into `PSUSP` state if your job is pending. Resuming a user suspended job does not put your job into `RUN` state immediately. If your job was running before the suspension, `bresume` first puts your job into `SSUSP` state and then waits for `sbatchd` to schedule it according to the load conditions.

Controlling Job Arrays

Each element of a job array is run independently of the others. You can kill, suspend, or resume all elements of the array, or only selected ones.

UNIX You can send an arbitrary signal to all elements of the array, or only selected ones.

Using the job id of the array operates on the all elements of the array. Selecting particular elements to control requires appending the index specification after the job id. For example:

```
% bsub -J "myArray[1-50]" sleep 10
Job <212> is submitted to default queue.
```

```
% bstop 212
Job <212>: Operation is in progress
```

```
% bresume 212
Job <212>: Operation is in progress
```

```
% bstop "212[5]"
Job <212[5]> is being stopped
```

```
% bstop "212[40-50]"
Job <212[40]> is being stopped
...
Job <212[50]> is being stopped
```

Note

When sending a command which operates on several elements of the array, the change in the status of the elements may not show up immediately. The system ensures the operation takes place in the background while other requests are being serviced.

7 Tracking Batch Jobs

The job name can also be used in selecting elements of the array (e.g `bstop -J "myArray[40-50]"`). Since multiple job arrays may have the same job name the command will affect all arrays with the name "myArray".

Changing the queueing position of a job through the `bbot` and `btop` commands can be done on individual elements of an array, but cannot operate on the entire array. For example, `btop 212[5]"` to move the element with index 5 in the job array with ID 212 to the first queueing position.

Sending an Arbitrary Signal to a Job

To send an arbitrary signal to your job, use the `-s` option of the `bkill` command. You can specify either the signal name or the signal number. On most versions of UNIX, signal names and numbers are listed in the `kill(1)` or `signal(2)` manual page. On Windows NT, only customized applications will be able to process job control messages specified with the `-s` option.

```
% bkill -s TSTP 3421
Job <3421> is being signaled
```

This example sends the TSTP signal to job 3421.

Note

Signal numbers are translated across different platforms because different operating systems may have different signal numbering. The real meaning of a specific signal is interpreted by the machine from which the `bkill` command is issued. For example, if you send signal 18 from an SunOS 4.x host, it means SIGTSTP. If the job is running on an HP-UX, SIGTSTP is defined as signal number 25, so signal 25 is sent to the job.

Only the owner of a batch job or an LSF administrator can send signals to a job.

You cannot send arbitrary signals to a pending job; most signals are only valid for running jobs. However, LSF Batch does allow you to kill, suspend and resume pending jobs.

Moving Jobs Within and Between Queues

The `btop` and `bbot` commands move pending jobs within a queue. `btop` moves jobs toward the top of the queue, so that they are dispatched before other pending jobs. `bbot` moves jobs toward the bottom of the queue so that they are dispatched later. The default behaviour is to move the job as close to the top or bottom of the queue as possible. By specifying a position on the command line, you can move a job to an arbitrary position relative to the top or bottom of the queue.

The `btop` and `bbot` commands do not allow users to move their own jobs ahead of those submitted by other users; only the dispatch order of the user's own jobs is changed. Only an LSF administrator can move one user's job ahead of another.

Note

The `btop` and `bbot` commands have no effect on the job dispatch order when fairshare policies are used.

```
% bjobs -u all
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user2	RUN	normal	hostA	hostD	sleep 500	Oct 23 10:16
5309	user2	PEND	night	hostA		sleep 200	Oct 23 11:04
5310	user1	PEND	night	hostB		myjob	Oct 23 13:45
5311	user2	PEND	night	hostA		sleep 700	Oct 23 18:17

```
% btop 5311
```

Job <5311> has been moved to position 1 from top.

```
% bjobs -u all
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user2	RUN	normal	hostA	hostD	sleep 500	Oct 23 10:16
5311	user2	PEND	night	hostA		sleep 700	Oct 23 18:17
5310	user1	PEND	night	hostB		myjob	Oct 23 13:45
5309	user2	PEND	night	hostA		sleep 200	Oct 23 11:04

Note that user1's job is still in the same position on the queue. User2 cannot use `btop` to get extra jobs at the top of the queue; when one of his jobs moves up on the queue, the rest of his jobs move down.

7 Tracking Batch Jobs

The `bswitch` command switches pending and running jobs from queue to queue. This is useful if you submit a job to the wrong queue, or if the job is suspended because of the queue thresholds or run windows and you would like to resume the job.

```
% bswitch priority 5309
```

```
Job <5309> is switched to queue <priority>
```

```
% bjobs -u all
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user2	RUN	normal	hostA	hostD	sleep 500	Oct 23 10:16
5309	user2	RUN	priority	hostA	hostB	sleep 200	Oct 23 11:04
5311	user2	PEND	night	hostA		sleep 700	Oct 23 18:17
5310	user1	PEND	night	hostB		myjob	Oct 23 13:45

Job Modification

There is no “pre-submission” modification. Using the `bmod` command, jobs are modified after they have been submitted. This section discusses the following topics:

- Submitted Job Modification
- Dispatched Job Modification
- Job Array Modification

Submitted Job Modification

For submitted jobs in PEND state, the `bmod` command is used by the job owner and LSF administrator to modify command line parameters (see ‘*Submitting Batch Jobs*’ on page 89).

To replace the job command line the `-Z "newCommand"` command option is used. The following example replaces the command line option for job 101 with "myjob file":

```
% bmod -Z "myjob file" 101
```

To change a specific job parameter, use `bmod` with the `bsub` option used to specify the parameter. The specified options replace the submitted options. The following example changes the start time of job 101 to 2:00 a.m.:

```
% bmod -b 2:00 101
```

To reset an option to its default submitted value (undo a `bmod`), append the `n` character to the option name, and do not include an option value. The following example resets the start time for job 101 back to its original value:

```
% bmod -bn 101
```

Resource reservation can be modified after a job has been started to ensure proper reservation and optimal resource utilization.

Dispatched Job Modification

For dispatched (started) jobs, the `bmod` command is used by the job owner and LSF administrator to modify resource reservations (see *'Resource Reservation' on page 91*). A job is usually submitted with a resource reservation for the maximum amount required. This command is used to decrease the reservation, allowing other jobs access to the resource. The following example sets the resource reservation for job 101 to 25MB of memory and 50MB of swap space:

```
% bmod -R "rusage[mem=25:swp=50]" 101
```

Individual elements of a job array can be modified after the array is submitted to LSF Batch. For example, this enables individual elements to have different resource requirements or different dependency conditions to control scheduling behavior.

Job Array Modification

When a job array is submitted, all the elements (jobs) within the array share the same job ID and submission parameters (i.e., resource requirements and submission queue). The `bmod` command is used by the job owner and LSF administrator to change the resource requirements for individual jobs or the entire array. Use the `bswitch` command to change the submission queue for individual jobs or the entire array. Both commands use the `jobId indexList` extension to support job array modifications.

7 Tracking Batch Jobs

Note

Job array modifications affect only those jobs that have not been dispatched. To make sure the modifications are applied to all specified jobs:

1. *Submit the job array on hold using the `-H` option, `bsub -H ...`*
2. *Make the job modifications, `bmod`, `bswitch`, ...*
3. *Release the job, `bresume jobId ...`*

Syntax

```
% bmod modification "jobId[indexList]"
```

```
% bswitch fromQ toQ "jobId[indexList]"
```

`modification`

specifies the resource modification using correct `bsub` syntax.

`fromQ, toQ`

specifies the queue to which the job was originally submitted, and the queue to which the job is to be switched.

`jobId`

specifies the job ID of the job array. The double quotes are not required if `indexList` does not follow the job ID.

`indexList`

specifies the elements (jobs) to be modified. Elements do not need to occupy continuous indices.

Examples

```
% bmod -R "mem >= 200" 101
```

changes the memory requirements for all jobs in the job array to 200MB

```
% bmod -R "mem >= 500" "101[3, 7]"
```

changes the memory requirements of jobs 3 and 7 to 500MB

```
% bmod -w "101[1]" "101[10]"
      makes sure job 10 runs after job 1
```

```
% bswitch defaultQ priorityQ "101[5]"
      changes the submission queue for job 5 from defaultQ to priorityQ
```

To replace the entire job command line after submission, use `bmod` with the `-Z` option, which takes the form `bmod -Z "new_command" jobId`. Consider the following example:

```
% bmod -Z "myjob file1" 12223
```

This command modifies the command for job 12223, changing it to “myjob file1”.

To change specific job parameters after submission, use `bmodify` with the option(s) you want to change. The `bmodify` command takes the same options as the `bsub` command together with a job ID (see ‘*Submitting Batch Jobs*’ on page 89). The given options replace the existing options of the specified job. For example, the following command changes the start time of job 123 to 2:00 a.m.

```
% bmod -b 2:00 123
```

To reset an option to its default value, append the `n` character to the option name, and do not include an option value. For example:

```
% bmod -bn 123
```

Job 123 will be dispatched as soon as possible, ignoring any previously specified start time.

Job arrays can be modified in the same way. Since all jobs share the same set of parameters, modifying the array will affect all jobs in the array.

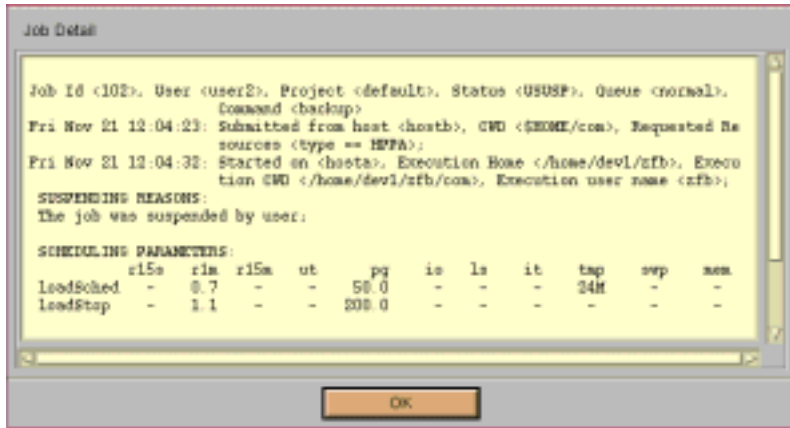
Job Tracking and Manipulation Using the GUI

Most of the operations discussed in this chapter can also be performed using the GUI. The main window of `xlsbatch` is shown in *Figure 4* on page 24.

7 Tracking Batch Jobs

You can view job details by first select a job and then click on the 'Detail' button. The resulting popup window is shown in *Figure 12*. This gives you the same information as you can get by running the `bjobs -l` command.

Figure 12. Detailed Job Information Popup Window



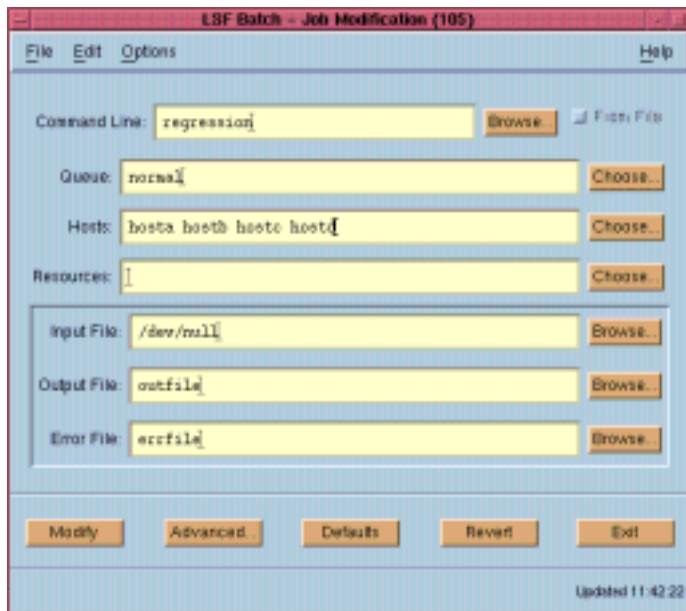
The 'History' button gives you a popup window for job history as you can otherwise get through the `bhist` command.

To perform control actions on jobs, such as killing a job or suspending/resuming a job, simply select the job and then click on an action button.

You can also invoke the `xbsub` window from inside `xlsbatch` to submit new jobs. If you want to modify a job parameter, simply select on the job and click on 'Modify' button to get the job modification popup window. Note that this window can also be invoked

by running `xbmod` from the command line. *Figure 13* shows the `xbmod` window. This window is the similar to the `xbsub` window.

Figure 13. Job Modification Window



8. Running Interactive Jobs

Interactive jobs communicate with the user in real time. Programs like `vi` use a text-based terminal interface; Computer Aided Design and desktop publishing applications usually use a graphic user interface (GUI).

LSF can run interactive jobs on any host in the cluster transparently. All interaction through a terminal or GUI, including keyboard signals such as `CTRL-C`, work the same as if the job was running directly on the user's host.

LSF supports interactive jobs in two optional ways:

- Using `lstoools` programs, such as `lsrun`, `lsgrun`, etc.
- Using LSF Batch to run interactive jobs.

This chapter contains detailed descriptions of the basic LSF tools for running jobs on remote hosts as well as ways to run interactive jobs via LSF Batch.

Shared Files and User IDs

When LSF runs a task on a remote host, the task uses standard UNIX system calls to access files and devices. The user must have an account on the remote host. All operations on the remote host are done with the user's access permissions.

Tasks that read and write files are accessing the files on the remote host. For load sharing to be transparent, your files should be available on all hosts in the cluster using a file sharing mechanism such as NFS or the Andrew File System (AFS). When your files are available on all hosts in the cluster, you can run your tasks on any host without worrying about how your task will access files.

8 Running Interactive Jobs

LSF can operate correctly in cases where these conditions are not met, but the results may not be what you expect. For example, the `/tmp` directory is usually private on each host. If you copy a file into `/tmp` on a remote host, you can only read that file on the same remote host.

LSF can also be used when files are not available on all hosts. LSF provides the `lsrscp(1)` command to copy files across LSF hosts. You can use pipes to redirect the standard input and output of remote commands, or write scripts to copy the data files to the execution host.

Running Remote Jobs with `lsrun`

The `lsrun` command runs a job on a remote host. The default is to run the job on the host with the least CPU load (the lowest normalized CPU run queue length) and the most available memory. Command line arguments can be used to select other resource requirements or to specify the execution host. For example, to run `myjob` on the best available host, enter:

```
% lsrun myjob
```

LSF automatically selects a host of the same type as the local host, if one is available. By default the host with the lowest CPU and memory load is selected.

If you want to run `myjob` on a host with specific resources, you can specify the resource requirements using the `-R resreq` option to `lsrun`.

```
% lsrun -R "swap>=100 && cserver" myjob
```

This command runs `myjob` on a host that has resource `cserver` (see ‘*Getting Cluster Information*’ on page 13) and has at least 100 megabytes of virtual memory available.

You can also configure LSF to store the resource requirements of specific jobs, as described in ‘*Configuring Resource Requirements*’ on page 53. If you configure LSF with the resource requirements of your job, you do not need to specify the `-R resreq` argument to `lsrun` on the command line. If you do specify resource requirements on the command line, they override the configured resource requirements.

If you want to run your job on a particular host, use the `-m` option to `lsrun`:

```
% lsrun -m hostD myjob
```

When you run an interactive job on a remote host, you can use signals as if it were running locally. If your shell supports job control, you can suspend and resume the job and bring the job to background or foreground as if it were a local job.

UNIX

Some jobs, such as text editors, require special terminal handling. These jobs must be run using a pseudo-terminal so that the special terminal handling can be used over the network. The `-P` option to `lsrun` specifies that the job should be run using a pseudo-terminal:

```
% lsrun -P vi
```

To avoid typing in the `lsrun` command every time you want to execute a remote job, you can also use a shell alias or script to run your job.

For a complete description of the command line arguments, see the `lsrun(1)` manual page.

Running Parallel Jobs with `lsgrun`

The `lsgrun` command allows you to run the same task on many hosts, either one after another or in parallel. For example, to merge the `/tmp/out` file on hosts *hostA*, *hostD*, and *hostB* into a single file named *gout*, enter:

```
% lsgrun -m "hostA hostD hostB" cat /tmp/out >> gout
```

To remove the `/tmp/core` file on all three hosts, enter:

```
% lsgrun -m "hostA hostD hostB" -p rm -r /tmp/core
```

The `-p` option tells `lsgrun` that the task specified should be run in parallel. If the `-p` argument is not given, tasks are run on each host one after another. See `lsgrun(1)` for more details.

8 Running Interactive Jobs

The `lsgrun -f host_file` option reads the *host_file* file to get the list of hosts on which to run the task.

Load Sharing Interactive Sessions

There are different ways to use LSF to start an interactive session on the best available host.

Load Sharing Login

To login to the least loaded host, the simplest way is to use the `lslogin` command. `lslogin` automatically chooses the best host and does an `rlogin` to that host. With no argument, `lslogin` picks a host that is lightly loaded in CPU, has few login sessions, and is binary compatible with the current host.

If you want to log into a host with specific resources, use the `lslogin -R resreq` option.

```
% lslogin -R "solaris order[ls:cpu]"
```

This command opens a remote login to a host that has the `sunos` resource, few other users logged in, and a low `cpu` load level. This is equivalent to using `lsplace` to find the best host and then using `rlogin` to log in to that host:

```
% rlogin `lsplace -R "sunos order[ls:cpu]"`
```

Load Sharing X Sessions

If you are using the X Window System, you can start an `xterm` that opens a shell session on the best host by entering:

```
% lsrsh -c "xterm &"
```

In this example, no processes are left running on the local host. The `lsrsh` command exits as soon as the `xterm` starts, and the `xterm` on the remote host connects directly to the X server on the local host.

If you are using a PC as a desk top machine and are running an X-Window server on your PC, then you can start an X session on the least loaded machine. The following steps assume you are using eXceed from Hummingbird Communications:

- Click the `Xstart` icon in the `eXceed4` program group
- Choose 'REXEC (TCP/IP, ...)' as start method, program type is X window
- Set the host to be any server host in your LSF cluster, for example `hostA`
- Set the command line to be:

```
% lsrun sh -c "xterm -ls -display yourPC:0.0&"
```

Note

The '&' in this command line is important as it frees resources on the server hostA once the xterm is running.

- Set description to be 'Best'
- Choose the 'install' button in the 'Xstart' window. This installs 'Best' as an icon in the program group you choose (for example, `xterms`).

Now, by double clicking on the 'Best' icon you will get an `xterm` started on the least loaded host in the cluster and displayed on your screen.

An alternative to start an X session on the best host is to submit it as a LSF Batch job:

```
% bsub xterm
```

This starts an `xterm` on the least loaded host in the cluster and displays on your screen.

When you run X applications using `lsrun` or `bsub`, the environment variable `DISPLAY` is handled properly for you. It behaves as if you were running the X application on the local machine.

Command-Level Job Starters

Some jobs have to be started in a particular environment or require some type of setup to be performed before they are executed. In a shell environment, this situation is often handled by writing such preliminary procedures into a file that itself contains a call to start the desired job. This is referred to as a *wrapper*.

If you want to run an interactive job that requires this type of preliminary setup, LSF provides a *job starter* function at the command level. A command-level job starter allows you to specify an executable file which will run prior to the actual job, doing any necessary setup and running the job when the setup is complete.

If the environment variable `LSF_JOB_STARTER` is properly defined, the RES will invoke the job starter (rather than the job itself), supplying your commands as arguments.

UNIX

The job starter is invoked from within a Bourne shell, making the command-line equivalent:

```
/bin/sh -c "$LSF_JOB_STARTER command [argument ...]"
```

where 'command [argument ...]' are the command line arguments you specified in `lsrun`, `lsgrun`, or `ch`.

If you define the `LSF_JOB_STARTER` environment variable as follows:

```
% setenv LSF_JOB_STARTER "/bin/csh -c"
```

Then you run a simple C-shell job:

```
% lsrun "'a.out; echo hi'"
```

The following will be invoked to correctly start the job:

```
/bin/sh -c "/bin/csh -c 'a.out; echo hi'"
```

NT

The RES runs the job starter, passing it your commands as arguments:

```
LSF_JOB_STARTER command [argument ...]
```


If you define the `LSF_JOB_STARTER` environment variable as follows:

```
set LSF_JOB_STARTER=C:\cmd.exe /C
```

Then you run a simple DOS shell job:

```
C:\> lsrun dir /p
```

The following will be invoked to correctly start the job:

```
C:\cmd.exe /C dir /p
```

A Job Starter can also be defined at the queue level (see *'Queue-Level Job Starters' on page 84* for more information) using the `JOB_STARTER` parameter, although this can only be done by the LSF administrator. The queue-level job starter is functionally similar to the interactive job starter, although it uses a slightly different mechanism. It is primarily used to customize LSF for particular environments (for example, to support Atria ClearCase).

Interactive Batch Job Support

When you run interactive jobs using `lsrun`, `lsgrun`, etc., these utilities use LIM's simple placement advice for host selection. It is sometimes desirable from a system management point of view to control all workload through a single centralized scheduler, LSF Batch.

Since all interactive jobs submitted to LSF Batch are subject to policies of LSF Batch, your system will have better control. For example, your system administrator may dedicate two servers as interactive servers and disable interactive access to all other servers by defining a interactive queue that only uses the two interactive servers.

Running an interactive job through LSF Batch also allows you to take the advantage of the batch scheduling policy and host selection features for resource intensive jobs.

8 Running Interactive Jobs

To submit an interactive job, you should first find out which queues accept interactive jobs by running `bqueues -l` command. If the output of this command contains:

```
SCHEDULING POLICIES: NO_INTERACTIVE
```

then this is a batch only queue. If the output contains:

```
SCHEDULING POLICIES: ONLY_INTERACTIVE
```

then this is an interactive only queue. If none of the above is defined or “SCHEDULING POLICIES” is not in the output of the `bqueues -l` command, then both interactive and batch jobs are accepted by this queue.

You can use LSF Batch to submit an interactive job with the `bsub` command. Your job can be submitted so that all input and output are through the terminal that you used to type the command.

An interactive batch job is submitted by specifying the `-I` option of the `bsub` command. When an interactive job is submitted, a message is displayed while the job is awaiting scheduling. The `bsub` command will block until the job completes and no mail is sent to the user by default. A user can issue a `CTRL-C` at any time to effectively terminate the job. For example:

```
% bsub -I -q interactive -n 4,10 lsmake
<<Waiting for dispatch ...>>
```

would start LSF Make on 4 to 10 processors and display the output on the terminal.

It is possible to use the `-I` option together with the `-i`, `-o`, and `-e` options (see *‘Input and Output’ on page 90*) to selectively redirect the streams to a files. For example:

```
% bsub -I -q interactive -e job.err lsmake
```

would save the standard error stream in the `'job.err'` file, while standard input and output would come from the terminal.

For jobs requiring pseudo-terminal support, `bsub` supports `-Ip` and `-Is` options. See the `bsub(1)` man page for more details.

Shell Mode for Remote Execution

Shell mode support is provided for running interactive applications through the RES or through LSF Batch. Shell mode support is required for running interactive shells or applications that redefine `CTRL-C` and `CTRL-Z` keys (for example, `jove`). The `-S` option to `lrun`, `ch` or `lsgrun` creates the remote task with shell mode support. The default is not to enable shell mode support. The `-Is` option to `bsub` provides the same feature for interactive batch jobs.

9. Using `lstcsh`

This chapter describes `lstcsh`, an extended version of the `tcsh` command interpreter. The `lstcsh` interpreter provides transparent load sharing of user jobs.

This chapter is not a general description of the `tcsh` shell; only the load sharing features are described in detail.

`lstcsh` is a convenient way for you to take advantage of LSF. Your commands are sent transparently for execution on faster hosts to improve response time or you can run commands on remote hosts explicitly.

Resource requirements for specific commands can be configured using task lists; see ‘*Configuring Resource Requirements*’ on page 53. Remote execution is transparent to the user; keyboard signals such as `CTRL-Z` and `CTRL-C` are automatically sent to the remote task.

NT

Note

Interactive tasks, including `lstcsh`, are not supported on Windows NT.

Starting `lstcsh`

If you normally use some other shell, you can start `lstcsh` from the command line. Make sure that the LSF commands are in your `PATH` environment variable and then enter `lstcsh`. If you have a `.cshrc` file in your home directory, `lstcsh` reads it to set variables and aliases. Use the `exit` command to get out of `lstcsh`.

Using `lstcsh` as Your Login Shell

If your system administrator allows, you can use LSF as your login shell. The `/etc/shells` file contains a list of all the shells you are allowed to use as your login shell. The `chsh` command can set your login shell to any of those shells. If the `/etc/shells` file does not exist, you cannot set your login shell to `lstcsh`.

For example, `user3` can run the command:

```
% chsh user3 /usr/local/lsf/bin/lstcsh
```

The next time `user3` logs in, the login shell will be `lstcsh`.

If you cannot set your login shell using `chsh`, you can use one of the standard system shells to start `lstcsh` when you log in. The easiest way is to use `chsh` to set `/bin/sh` to be your login shell and then edit the `.profile` file in your home directory to start `lstcsh`, as shown below.

```
SHELL=/usr/local/lsf/bin/lstcsh
export SHELL
exec $SHELL -l
```

Automatic Remote Execution

Every time you enter a command, `lstcsh` looks in your task lists to determine whether the command can be executed on a remote host and to find the configured resource requirements for the command. See *'Configuring Resource Requirements'* on page 53 for more information about task lists.

If the command can be executed on a remote host, `lstcsh` calls the LIM to find the best available host. The first time a command is run on a remote host, a server shell is started on that host. The command is sent to the server shell, and the server shell starts the command on the remote host. All commands sent to the same host use the same server shell, so the start-up overhead is only incurred once.

If no host is found that meets the resource requirements of your command, it is run on the local host.

Host Redirection

You can explicitly specify the eligibility of a command line for remote execution using the '@' character. It may be anywhere in the command line except in the first position ('@' as the first character on the line is used to set the value of shell variables).

Host redirection overrides the task lists, so you can force commands from your local task list to execute on a remote host or override the resource requirements for a command.

'@' followed by nothing means that the command line is eligible for remote execution. '@' followed by a host name forces the command line to be executed on that host. '@' followed by the reserved word `local` forces the command line to be executed on the local host only. '@' followed by '/' and a resource requirement string means that the command is eligible for remote execution and that the given resource requirements must be used instead of those in the remote task list.

```
% hostname @hostD
<< remote execution on hostD >>
hostD
% hostname @/type==alpha
<< remote execution on hostB >>
hostB
```

For ease of use, the host names and the reserved word `local` following '@' can all be abbreviated as long as they do not cause ambiguity. Similarly, when specifying resource requirements following the '@', it is necessary to use '/' only if the first requirement characters specified are also the first characters of a host name.

You do not have to type in resource requirements for each command line you type if you put these task names into remote task list together with their resource requirements by running `lsrtasks`.

Job Control

Job control in `lstcsh` is the same as in `tcsh` except for remote background jobs. `lstcsh` numbers shell jobs separately for each execution host.

The output of the built-in command `jobs` lists the background jobs together with their execution hosts. This break of transparency is intentional to provide you with more control over your background jobs.

```
% sleep 30 @hostD &
<< remote execution on hostD >>
[1] 27568
% sleep 40 @hostD &
<< remote execution on hostD >>
[2] 10280
% sleep 60 @hostB &
<< remote execution on hostB >>
[1] 3748
% jobs
<hostD>
[1]  + Running          sleep 30
[2]   Running          sleep 40
<hostB>
[1]  + Running          sleep 60
```

To bring a remote background job to the foreground, the host name must be specified together with '@', as in the following example:

```
% fg %2 @hostD
<< remote execution on hostD >>
sleep 40
```

Built-in Commands

`lstcsh` supports two built-in commands to control load sharing, `lsmode` and `connect`.

The `lsmode` Command

The `lsmode` command takes a number of arguments that control how `lstcsh` behaves. With no arguments, `lsmode` displays the current settings:

```
% lsmode
LSF 3.1
Copyright 1992-1997 Platform Computing Corporation
LSF enabled, local mode, LSF on, verbose, no_eligibility_verbose, notiming.
```

The `lsmode` command reports that LSF is enabled if `lstcsh` was able to contact the LIM when it started up. If LSF is disabled, no load-sharing features are available.

`lsmode [on | off]`
Turns load sharing on or off. The default is on.

`lsmode [local | remote]`
Sets `lstcsh` to use local or remote mode. The default is local. Refer to '*Modes of Operation*' on page 154 for a description of local and remote modes.

`lsmode [e | -e]`
Turns on (e) or off (-e) eligibility verbose mode. If eligibility verbose mode is on, `lstcsh` shows whether the command is eligible for remote execution, and displays the resource requirement used if the command is eligible. The default is off.

`lsmode [v | -v]`
Turns on (v) or off (-v) task placement verbose mode. If verbose mode is on, `lstcsh` displays the name of the host where the command is run, if the command is not run on the local host. The default is on.

`lsmode [t | -t]`
Turns on (t) or off (-t) wall clock timing. If timing is on, the actual response time of the command is printed. This time includes all remote execution overhead. The default is off.

The `connect` Command

`lstcsh` opens a connection to a remote host when the first command is executed remotely on that host. The same connection is used for all future remote executions on that host. The `lstcsh connect` command with no argument displays the connections that are currently open.

The `connect host` command creates a connection to the named host. By connecting to a host before any command is run, the response time is reduced for the first remote command sent to that host.

`lstcsh` has a limited number of ports available to connect to other hosts. By default each shell can only connect to 15 other hosts.

```
% connect
CONNECTED WITH          SERVER SHELL
hostA                    +
```

```
% connect hostB
Connected to hostB
```

```
% connect
CONNECTED WITH          SERVER SHELL
hostA                    +
hostB                    -
```

The server shell is a process created on the remote host to handle `lstcsh` jobs. The server shell is started when the first command is executed on the remote host. In the previous example, the `connect` command created a connection to host `hostB`, but the server shell has not started.

Modes of Operation

LSF maintains two task lists for each user, a local list and a remote list. Commands in the local list must be executed locally. Commands in the remote list can be executed remotely. If a command is in neither list, you can choose how `lstcsh` handles the command.

`lstcsh` has two modes of operation: *local* and *remote*. The local mode is the default mode. In local mode, a command line is eligible for remote execution only if all of the commands on the line are present in the remote task list, or if the '@' character is specified on the command line to force it to be eligible. In remote mode, a command line is considered eligible for remote execution, if none of the commands on the line is in the local task list.

Local mode is conservative and can fail to take advantage of the performance benefits and load-balancing advantages of LSF. Remote mode is aggressive and makes more extensive use of LSF. However, remote mode can cause inconvenience when `lstcsh` tries to send host-specific commands to other hosts.

Using the LSF commands `lsltasks(1)` and `lsrtasks(1)`, you can inspect and change the memberships of the local and remote task lists. You can optionally associate resource requirements with each command in the remote list to help LSF find a suitable execution host for the command. If there are multiple eligible commands on a command line, their resource requirements are combined for host selection. See *'Remote Task List File' on page 53* for more information on using task lists and resource requirements.

Differences from Other Shells

When a command is running in the foreground on a remote host, all keyboard input (type-ahead) is sent to the remote host. If the remote command does not read the input, it is lost. `lstcsh` has no way of knowing whether the remote command reads its standard input. The only way to provide any input to the command is to send everything available on the standard input to the remote command in case the remote command needs it. As a result, any type-ahead entered while a remote command is running in the foreground, and not read by the remote command, is lost.

The '@' character has a special meaning when it is preceded by white space. This means that the '@' must be escaped with a backslash '\' to run commands with arguments that start with '@', like `finger`. This is an example of using `finger` to get a list of users on another host:

```
% finger @other.domain
```

Normally the `finger` command tries to contact the named host. Under `lstcsh`, the '@' character is interpreted as a request for remote execution, so the shell tries to contact the RES on the host *other.domain* to remote execute the `finger` command. If this host is not in your LSF cluster, the command fails. When the '@' character is escaped, it is passed to `finger` unchanged and `finger` behaves as expected.

```
% finger \@hostB
```

Writing Shell Scripts in `lstcsh`

You should write shell scripts in `/bin/sh` and use the `lstools` commands for load sharing. However, `lstcsh` can be used to write load-sharing shell scripts.

By default, an `lstcsh` script is executed as a normal `tcsh` script with load-sharing disabled. The `lstcsh -L` option tells the `lstcsh` that the script should be executed with load sharing enabled, so individual commands in the script may be executed on other hosts.

There are three different ways to run an `lstcsh` script with load sharing enabled: run `lstcsh -L script`, start an interactive `lstcsh` and use the `source` command to read the script in, or make the script executable and put `'#! /usr/local/lsf/bin/lstcsh -L'` as the first line of the script (assuming you install `lstcsh` in the `/usr/local/lsf/bin` directory).

Limitations

A shell is a very complicated application by itself. `lstcsh` has certain limitations:

- Native Language System is not supported. To use this feature of the `tcsh`, you must compile `tcsh` with `SHORT_STRINGS` defined. This causes complications for characters flowing across machines.

-
- Shell variables not propagated across machines. When you set a shell variable locally, then run a command remotely, the remote shell will not see that shell variable. Only environment variables are automatically propagated.
 - The `fg` command for remote jobs must use '@', as shown by examples in *'Job Control'* on page 152.
 - `lstcsh` is based on `tcsh 6.03 (7 bit mode)`. It does not support the new features of the latest `tcsh`.

10. Using LSF Make

This chapter describes how to use LSF Make to perform parallel software builds and similar tasks.

LSF Make is based on GNU `make`, and supports all GNU `make` features. Additional command line options control parallel execution. GNU `make` is upwardly compatible with the `make` programs supplied by most UNIX vendors.

To use LSF Make you do not need to change your makefile, although reorganizing the contents of the makefile might increase the parallelism and therefore reduce the running time.

The `lsmake(1)` manual page describes the command line options that control load sharing. The `gmake(1)` manual pages describes the other command line options.

NT

Note

LSF Make is not supported on Windows NT.

Parallel Execution

Many tasks consist of many subtasks, with dependencies between the subtasks. For example, compiling a software package requires compiling each file in the package and then linking all the compiled files together.

In many cases most of the subtasks do not depend on each other. For a software package, the individual files in the package can be compiled at the same time; only the linking step needs to wait for all the other tasks to complete.

In an LSF cluster you can use LSF Make to select a group of hosts and run parts of your make in parallel.

Invoking LSF Make

LSF Make supports all the GNU `make` command line options.

Specifying the Number of Processors

The `lsmake -j num_processors` option tells LSF Make to ask the LIM for *num_processors* processors. If fewer processors are available, LSF Make uses all the available processors. If no processors are available and the local host meets all resource requirements specified using the `lsmake -R` option, all `make` tasks are run on the local host.

By default LSF Make selects the same host type as the submitting host. This is necessary for most compilation jobs; all components must be compiled on the same host type and operating system version to run correctly. If your `make` task requires other resources you can override the default resource requirements with the `lsmake -R resreq` option.

For example, to build your software in parallel on 10 processors, enter:

```
% lsmake -j 10
```

If you want to take advantage of parallelism between the CPU and I/O on a powerful host, you can also specify the number of concurrent jobs for each processor using the `lsmake -c` option, as follows:

```
% lsmake -j 10 -c 2
```

This selects up to 10 processors and starts two tasks on each processor.

File Server Load

LSF Make can significantly reduce the response time of your `make`; however, it may also overload your file server or network if the jobs you are running are I/O intensive.

You can specify a threshold load so that parallelism is automatically reduced when the file server load is above a threshold and expanded when the file server load is below the threshold.

```
% lsmake -j 10 -F "r15s < 5 && pg < 20"
```

This LSF Make job uses up to 10 hosts, and reduces the parallelism if the file server CPU load `r15s` goes beyond 5, or if the file server paging rate goes beyond 20 pages per second. LSF Make automatically determines the file server for the current working directory.

Tuning Your Makefile

The smallest unit that LSF Make runs in parallel is a single `make` rule. If your makefile has rules that include many steps, or rules that contain shell loops to build sub-parts of your project, LSF Make runs the steps serially.

You can increase the parallelism in your makefile by breaking up complex rules into groups of simpler rules. Steps that must run in sequence can use `make` dependencies to enforce the order. LSF Make can then find more subtasks to run in parallel.

Building in Subdirectories

If your `make` job is divided into subdirectories, `lsmake -M` can process the subdirectories in parallel. The total number of parallel tasks is shared over all the subdirectories. Without the `-M` option, LSF Make processes subdirectories sequentially, although tasks within each subdirectory can be run in parallel.

To process subdirectories in parallel they must be built as separate targets in your makefile. You must specify the `make` command for each subdirectory with the built-in `$(MAKE)` macro so that LSF Make can substitute the correct `lsmake` command for the subdirectory.

10 Using LSF Make

Some makefiles may work correctly when run on a single machine, but may not work correctly when run in parallel through LSF Make.

Below is a makefile rule that uses a shell loop to process subdirectories.

```
DIRS = lib misc main
prog:
    for subdir in $(DIRS) ; do \
        cd $subdir ; $(MAKE) ; cd .
```

When this makefile is run on a single machine, the directories are processed sequentially, i.e. `lib` is built before `misc` and `main`. However, when run using `lsmake` with the `-M` option, all directories can be built in parallel. Therefore, it is possible for the `misc` and `main` directories to be built before `lib`, which is not correct.

Below is a set of makefile rules to perform the same tasks and allows the subdirectories to be built in parallel in the correct order. An extra rule is added so that the `lib` and `misc` subdirectories are built before the main directory:

```
DIRS = lib misc main
prog: $(DIRS)
$(DIRS):
    cd $@ ; $(MAKE)
```

Running `lsmake` as a Batch Job

`make` jobs often require a lot of resources, but no user interaction. Such jobs can be submitted to the LSF Batch system so that they are processed when the needed resources are available. `lsmake` includes extensions to run as a parallel batch job under LSF Batch:

```
% bsub -n 10 lsmake
```

This command queues an LSF Make job that needs 10 hosts. When all 10 hosts are available, LSF Batch starts LSF Make on the first host, and passes the names of all hosts in an environment variable. LSF Make gets the host names from the environment variable and uses the RES to run tasks.

You can also specify a minimum and maximum number of processors to dedicate to your `make` job (see ‘*Minimum and Maximum Number of Processors*’ on page 104):

```
% bsub -n 6,18 lsmake
```

Because LSF Make passes the suspend signal (SIGTSTP) to all its remote processes, the entire parallel `make` job can be suspended and resumed by the user or the LSF Batch system.

Differences from Other Versions of `make`

LSF Make is fully compatible with GNU `make`. There are some incompatibilities between GNU `make` and some other versions of `make`; these are beyond the scope of this document.

When LSF Make is running processes on more than one host, it does not send standard input to the remote processes. Most makefiles do not require any user interaction through standard I/O. If you have makefile steps that require user interaction, you can put the commands that require interaction into your local task list. Commands in the local task list always run on the local host, where they can read from standard input and write to standard output.

11. Checkpointing and Migration

Checkpointing allows you to take a snapshot of the state of a running batch job, so that the job can be restarted later. There are two main reasons for checkpointing a job: fault tolerance and load balancing.

A batch job can be checkpointed periodically during its run. If the execution host goes down for any reason, the job can resume execution from the last checkpoint when the host comes back up, rather than having to start from the beginning. The job can also be restarted on a different host if the original execution host remains unavailable.

Sometimes one host is overloaded while the others are idle or lightly loaded. LSF Batch can checkpoint one or more jobs on the overloaded host and restart the jobs on other idle or lightly loaded hosts. Job migration can also be used to move intensive jobs away from hosts with interactive users such as desktop workstations, so that the intensive jobs do not interfere with the users.

Topics covered in this chapter are:

- approaches to checkpointing
- submitting a checkpointable job
- checkpointing a job
- restarting a job from its checkpoint
- job migration
- submitting jobs that are automatically rerun if the execution host fails
- building checkpointable batch jobs

Approaches to Checkpointing

Checkpointing can be implemented at three levels. These levels are called *kernel-level*, *user-level* and *application-level*.

LSF Batch uses the kernel-level support available on ConvexOS, Cray (UNICOS), and SPP-UX, IRIX 6.4, and NEC SX-4 systems to implement checkpointing on these machines. LSF Batch provides user-level support on most other platforms (HP-UX, DEC Alpha, SGI IRIX 5.3 and 6.2, Solaris 2.5.1 and 2.6, SunOs 4, and AIX 4). LSF Batch also supports application-level checkpointing.

Kernel-level Checkpointing

In kernel-level checkpointing, the operating system supports checkpointing and restarting processes. The checkpointing is transparent to applications: they do not have to be modified or linked with any special library to support checkpointing.

User-level Checkpointing

Application programs to be checkpointed are linked with a checkpoint user library. Upon checkpointing, a checkpoint triggering signal is sent to the process. The functions in the checkpoint library respond to the signal and save the information necessary to restart the process. On restart, the functions in the checkpoint library restore the execution environment for the process. To applications, checkpointing is transparent. But unlike kernel-level support, applications must be relinked to allow checkpointing.

Application-level Checkpointing

Applications can be coded in a way to checkpoint themselves either periodically or in response to signals sent by other processes. When restarted, the application must look for the checkpoint files and restore its state.

Checkpoint Directory

Checkpoint directory stores checkpoint files that are necessary to recreate a checkpointed job. LSF allows more than one job to be checkpointed into the same directory. Each job is saved in a different subdirectory named by the job ID.

LSF Batch automatically renames `chkpntdir/jobId` to `chkpntdir/newjobId` after a job is restarted. A conflict may occur when a directory `chkpntdir/jobId` already exists and the restart jobID is the same as the existing directory. In this case, LSF Batch first renames the existing directory `chkpntdir/jobId` to `chkpntdir/jobId.bak`.

If the conflict cannot be solved, `sbatchd` will report it to `mbatchd` and the job will be aborted.

Uniform Checkpointing Interface

All interaction between LSF and the platform or application dependent facility goes through a common interface provided by two executables: `echkpnt` and `erestart`¹. `echkpnt` is invoked when LSF needs to checkpoint a job. `erestart` is invoked when LSF needs to start a previously checkpointed job.

For systems supporting kernel-level checkpointing, `echkpnt` and `erestart` are just wrapper scripts for the vendor's checkpointing commands. User-level checkpointing for the supported platforms uses whichever `echkpnt/erestart` interacts with the supplied checkpoint library.

You can follow the format given below to extend or replace `echkpnt` and `erestart` for other checkpoint facilities, including the application-level checkpointing.

The `echkpnt` Command

The `echkpnt` command must support the following arguments and take appropriate actions to perform the checkpointing:

```
echkpnt [-c] [-f] [-k | -s] [-d chkpnt_dir ] process-group-id
```

1. `echkpnt` and `erestart` are by default located in the `LSF_SERVERDIR` directory (as defined in `lsf.conf`). You can specify another directory with the `LSF_ECHKPNTDIR` environment variable.

11 Checkpointing and Migration

- c** Copy all regular files in use by the checkpointed process to the checkpoint directory.
 - f** Checkpoint the job even if non-checkpointable conditions exist (non-checkpointable conditions are specific to the type of checkpoint facility being used). This may create checkpoint files that will not restart properly.
 - k** Kill the job if the checkpoint operation is successful. Default is that the job continues execution after being checkpointed. If this option is specified and the checkpoint fails for any reason, the job continues normal execution.
 - s** Stop the job if the checkpoint operation is successful. Default is that the job continues execution after being checkpointed. If this option is specified and the checkpoint fails for any reason, the job continues normal execution.
- d `chkpnt_dir`**
The directory containing the checkpoint files. Default is the current directory.

`process-group-id`
The process group ID of the job to be checkpointed.

The `erestart` Command

The `erestart` command must support the following arguments and take appropriate actions to perform the restart:

`erestart [-c] [-f] chkpnt_dir`

- c** Copy data files from the checkpoint directory to the original pathname. If any of the process's data files are copied into the checkpoint directory at the time of checkpoint, this option will cause restart to replace the contents of the original files with the copies. This option is currently only supported on ConvexOS.
- f** Force a restart of the process even if non-restartable conditions exist.

`chkpnt_dir`
Checkpoint directory.

When LSF Batch calls `erestart`, it will first pass the process-ID and process group-ID of the original job via the environment variables `LSB_RESTART_PID` and `LSB_RESTART_PGID`, respectively. Then it waits for a message from the standard error of `erestart` before proceeding. Upon success, `erestart` sends back either a message of "`pid=NEW_PID pgid=NEW_PGID`" if there are any restart process-id or process group-id changes, or a null message by closing the standard error if there are not any changes. On failure, LSF Batch expects `erestart` will write the error messages via its standard error.

Submitting Checkpointable Jobs

Checkpointable jobs are submitted using the `-k "checkpoint_dir[period]"` option of the `bsub` command. The `checkpoint_dir` parameter specifies the directory where the checkpoint files are created. The `period` parameter specifies an optional time interval in minutes for automatic and periodic checkpointing. If the `period` parameter is specified, the checkpoint directory and period must be enclosed in quotes.

If the checkpoint period is not specified, the job is considered checkpointable but it is not automatically checkpointed. A checkpoint can be created for any checkpointable batch job using the `bchkpnt` command. If the checkpoint period is specified, LSF Batch automatically checkpoints the job at the specified time interval.

```
% bsub -k "io.chkdir 10" io_job
Job <3426> is submitted to default queue <normal>.
```

```
% bjobs -l
```

```
Job Id <3426>, User <user2>, Status <RUN>, Queue <normal>, Command <io_job>
Thu Oct 24 16:50:27: Submitted from host <hostB>, CWD <$HOME/tmp>, Checkpoint period 10 min., Checkpoint directory <io.chkdir/3426>;
Thu Oct 24 16:50:28: Started on <hostB>;
```

	r15s	rlm	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	0.7	1.0	-	4.0	-	12	-	0M	-	-
loadStop	-	1.5	2.5	-	8.0	-	15	-	-	-	-

11 Checkpointing and Migration

This example submits the checkpointable batch job `io_job`. Checkpoint files will be stored in the directory `io.chkdir/3426` (relative to the current directory), and checkpoints will automatically be created every 10 minutes, each overwriting the previous one.

Note

It is your responsibility to clean up the checkpoint directory when it is no longer needed.

If the checkpoint directory does not exist, LSF Batch creates it. If it does exist, it must be writable by the user. To restart a job on a different host, the checkpoint files must be stored in a directory accessible to both hosts.

Checkpointing a Job

In addition to automatic checkpointing, users can checkpoint jobs explicitly with the `bchkpnt` command. A job is checkpointable only if it was submitted with the `-k` option to `bsub`.

The `-p period` option to `bchkpnt` allows you to set or change the checkpointing period for a job. *period* is specified in minutes. If *period* is 0 (zero), periodic checkpointing is turned off. Otherwise, periodic checkpointing is turned on and the checkpoint period is set to the value given.

Some jobs cannot be checkpointed and restarted correctly because it is not possible to recreate the running state of the job. See ‘*Limitations*’ on page 178 and the `chkpnt(1)` manual page on ConvexOS and Cray systems for a list of conditions that can cause checkpointing to fail.

The `-f` option to `bchkpnt` forces the job to be checkpointed, even if some condition exists that would normally make the job non-checkpointable. When a non-checkpointable job is checkpointed using the `-f` flag, the job may not be restarted correctly.

The `-k` option to `bchkpnt` checkpoints and kills the batch job as an atomic action. This guarantees that the job does not do any processing or I/O after the checkpoint, so that the restarted job does not repeat any operations already performed by the original job.

```
% bchkpnt -f -p 15
```

```
Job <3426> is being checkpointed
```

```
% bjobs -l
```

```
Job Id <3426>, User <user2>, Status <RUN>, Queue <normal>, Command <io_job>
Thu Oct 24 16:50:27: Submitted from host <hostB>, CWD <${HOME}/tmp>, Checkpoint period 15 min., Checkpoint directory <io.chkdir/3426>;
Thu Oct 24 16:50:28: Started on <hostB>;
```

	r15s	rlm	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched -	0.7	1.0	-	4.0	-	12	-	0M	-	-	-
loadStop -	1.5	2.5	-	8.0	-	15	-	-	-	-	-

```
% bhist -l
```

```
Job Id <3426>, User <user2>, Command <io_job>
Thu Oct 24 16:50:27: Submitted from host <hostB> to Queue <normal>, CWD <${HOME}/tmp>, Checkpoint period 10 min., Checkpoint directory <io.chkdir/3426>;
Thu Oct 24 16:50:28: Started on <hostB>, Pid <4705>;
Thu Oct 24 16:51:21: Checkpoint initiated (actpid 4717);
Checkpoint period is 15 min.;
Thu Oct 24 16:51:29: Checkpoint succeeded (actpid 4717).
```

```
Summary of time in seconds spent in various states by Fri Oct 24 16:51:36 1997
```

PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
1	0	68	0	0	0	69

Restarting a Checkpointed Job

A checkpointed batch job can be restarted on a host of the same type as, and running the same operating system version as, the original execution host. The executable being run and all open files must be located at the same absolute path name.

In addition to automatic job restart, specified with the `-r` option to `bsub`, LSF Batch provides the `brestart` command that restarts a checkpointed batch job from the

11 Checkpointing and Migration

information stored in the checkpoint directory. The checkpoint directory must have been created by a previously submitted job that was checkpointed successfully.

When you restart a job with `brestart` command, you should specify a *jobId* and *chkpntDir* directory. The command looks like:

```
brestart  chkpntDir  [lastJobId]
```

where *lastJobId* must be specified if there is more than one *jobId* directory under *chkpntDir*.

`brestart` creates a new batch job that uses the checkpoint information. The restarted job is assigned a new job ID number, and is placed at the end of the specified queue. The job begins executing when a suitable host is available, like any other batch job.

The `brestart` command takes many of the same options as the `bsub` command to specify the conditions under which the restarted job runs. The restarted job has the same output file and file transfer specifications, job name, run window signal value, checkpoint directory and period, and rerunability as the original job.

```
% brestart io.chkdir 3426
```

```
Job <3427> is submitted to default queue <normal>.
```

```
% bjobs -l
```

```
Job Id <3427>, User <user2>, Status <RUN>, Queue <normal>, Command <io_job>
```

```
Thu Oct 24 17:05:01: Submitted from host <hostB>, CWD <$HOME/tmp>, Checkpoint directory <io.chkdir/3427>;
```

```
Thu Oct 24 17:05:01: Started on <hostB>;
```

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	0.7	1.0	-	4.0	-	12	-	0M	-	-
loadStop	-	1.5	2.5	-	8.0	-	15	-	-	-	-

Job Migration

Checkpointable jobs and rerunnable jobs (jobs that are submitted with the `bsub -r` option) can be migrated to another host for execution if the current host is too busy or the host is going to be shut down. Such jobs can be moved from one host to another, as long as both hosts are binary compatible and run the same version of the operating system.

The job's owner or the LSF administrator can use the `bmig` command to migrate jobs. If the job is checkpointable, the `bmig` command first checkpoints it. Then LSF kills the running or suspended job, and restarts or reruns the job on another suitable host if one is available. If LSF is unable to rerun or restart the job immediately, the job reverts to `PEND` status and is requeued with a higher priority than any submitted job, so it is rerun or restarted before other queued jobs are dispatched.

```
% bmig 3426
```

```
Job <3426> is being migrated
```

```
% bhist -l 3426
```

```
Job Id <3426>, User <user2>, Command <io_job>
```

```
Thu Oct 24 16:50:27: Submitted from host <hostB> to Queue <normal>, CWD <$HOME/tmp>, Checkpoint period 10 min., Checkpoint directory <io.chkdir/3426>;
```

```
Thu Oct 24 16:50:28: Started on <hostB>, Pid <4705>;
```

```
Thu Oct 24 16:51:21: Checkpoint initiated (actpid 4717);
```

```
Checkpoint period is 15 min.;
```

```
Thu Oct 24 16:51:29: Checkpoint succeeded (actpid 4717);
```

```
Thu Oct 24 16:53:42: Migration requested;
```

```
Thu Oct 24 16:54:03: Migration checkpoint initiated (actpid 4746);
```

```
Thu Oct 24 16:54:15: Migration checkpoint succeeded (actpid 4746);
```

```
Thu Oct 24 16:54:15: Pending: Migrating job is waiting for reschedule;
```

```
Thu Oct 24 16:55:16: Started on <lemon>, Pid <10354>.
```

```
Summary of time in seconds spent in various states by Fri Oct 24 16:57:26 1997
```

PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
62	0	357	0	0	0	419

Queues and Hosts for Automatic Job Migration

LSF Batch will not automatically migrate a job unless the job has a migration threshold. A job has a migration threshold if it is dispatched from a queue or to a host that has a migration threshold defined. The `bqueues -l` and `bhosts -l` commands display the migration threshold if it is defined.

Automatically Rerunning and Restarting Jobs

Batch jobs submitted with the `-r` option to the `bsub` command are automatically rerun or restarted if the execution host becomes unavailable. (A host is considered unavailable if both the LIM and `sbatchd` daemons are unreachable.) If the job has not been checkpointed, the job is rerun from the beginning. If the job has been checkpointed, either automatically or by the `bchkpnt` command, it is restarted from the last checkpoint.

When a job is rerun or restarted, it is returned to the batch queue in which it was executing, with the same options but a higher priority as the original batch job. The job uses the same job ID number. It is executed when a suitable host is available, and an email message is sent to the job submitter informing the user of the restart.

Submitting a Job for Automatic Migration

If you want to submit a rerunnable job and have the system automatically migrate it to another host when the job is suspended due to load, you must submit the job to a queue or a host configured for automatic job migration.

If a job with a migration threshold has been suspended for more than the specified number of minutes, LSF Batch attempts to migrate the job to another host. Only checkpointable or rerunnable jobs are considered for migration. To submit a rerunnable job, you must use the `-r` option to the `bsub` command.

If you want a job to be rerunnable but you do not want the system to migrate it automatically, submit the job to a queue or host that does not have a migration threshold defined. You can still migrate the job manually with the `bmig` command.

Building Checkpointable Jobs

The LSF checkpoint library provides user-level checkpointing facilities on operating systems that do not provide kernel checkpointing. It consists of three parts:

- the checkpoint library, `libckpt.a`
- the special checkpoint startup routine, `ckpt crt0.o`
- the checkpoint linkers, `ckpt_ld` and `ckpt_ld_f`

Programs to be checkpointed must be linked with the checkpoint startup routine and library. The checkpoint linkers are shell scripts that call the standard linkers on your operating system with the correct options to link a checkpointable program.

The LSF user-level checkpoint library is based on the Condor system from the University of Wisconsin.

The Checkpoint Library

The checkpoint library consists of a set of file system call stubs for file operations such as `open()`, `close()`, and `dup()`, intercepted by the checkpoint library. It also contains a checkpoint signal handler and routines used internally to implement checkpointing.

The Checkpoint Startup Routine

The startup routine sets the checkpoint signal handler and initializes some data structures. The data structures are used to record file accesses of the process and are used during the restart to re-open files opened before the checkpoint and to restore the current access positions in those files. The signal handler settings of the process are also saved at the checkpoint time and restored when the process is restarted. All these operations are transparent to applications.

Linking

Except on those systems with kernel checkpointing support, checkpointable programs must be linked with the special checkpoint library and startup routine instead of the

11 Checkpointing and Migration

standard ones. LSF comes with replacement linkers for programs to be checkpointed. The `ckpt_ld` and `ckpt_ld_f` commands are shell scripts that take the same parameters as the system linker, `ld`. The checkpoint linkers call `ld` with the correct flags to link the user's program with the checkpoint library `libckpt.a` and the special startup routine `ckpt_crt0.o`. `ckpt_ld` is for linking C programs and `ckpt_ld_f` is for linking FORTRAN programs.

In most cases, users run the compiler for the language used, and the compiler then invokes the linker on behalf of the user. Because of this, few users call the linker directly. This section describes the steps involved in building a checkpointable application.

Suppose you have a C program called `prog.c`, and you normally create an executable, `prog`, as follows:

```
% cc -o prog prog.c
```

To build a checkpointable program, however, you need to build the object file, `prog.o`, as follows:

```
% cc -c prog.c
```

Note

On SGI systems running IRIX version 5 or 6, use the `-nonshared` option.

Use the LSF linker script to build the executable:

```
% ckpt_ld -o prog prog.o
```

Note

ckpt_ld requires the static library `libc.a` on all platforms. In addition, on AIX version 3.2 it requires `libbsd.a`; on Solaris version 2 it requires `libmalloc.a`, `libsocket.a`, `libnsl.a` and `libintl.a`.

Like C programs, FORTRAN source files must be compiled to object files and then linked with the `ckpt_ld_f` command. To build a checkpointable version of `prog1` from the `prog1.f` source file, proceed as follows:

```
% f77 -c prog1.f
```


Note

On SGI systems running IRIX version 5 or 6, use the `-nonshared` option.

```
% ckpt_ld_f -o prog1 prog1.o
```

Note

ckpt_ld_f requires the static library `libc.a` on all platforms. It requires additional static libraries on each platform, as shown in Table 7.

Table 7. Static Libraries for `ckpt_ld_f`

Platform	Static libraries (lib*.a)
ULTRIX 4	Ufor, for, i, m, ots, util
Digital Unix	Futil, Ufor, for, m, ots
HP-UX 9 HP-UX 10	F77, I77, U77, cl, isamstub, utchem
AIX 3.2	bsd, m, xlf
AIX 4.1	m, xlf
IRIX 5 IRIX 6, mips 2	F77, I77, U77, isam, m
IRIX 6, mips 3 IRIX 6, mips 4	ftn, m
SunOS 4	F77, ansi, m
Solaris 2	F77, intl, m, malloc, nsl, socket, sunmath

In addition to the checkpoint library and startup routine, `ckpt_ld` and `ckpt_ld_f` may also link in a few other system-provided object files that are platform-dependent. Normally, these files are installed in a standard directory by the operating system. However, you may get an error if your system administrator set things up differently. In this case, see the `ckpt_ld(1)` manual page for more information.

Limitations

There are restrictions to the use of the current implementation of the checkpoint library for user level checkpointing. These are:

- The checkpointed process can only be restarted on hosts of the same architecture and running the same operating system as the host on which the checkpoint was created.
- Only single process jobs can be checkpointed.
- Processes with open pipes or sockets can be checkpointed but may not properly restart as the pipes/sockets are not re-opened on restart.
- If a process has `stdin`, `stdout`, or `stderr` as open pipes, all data in the pipes is lost on restart.
- The checkpointed process cannot be operating on a private stack when the checkpoint happens.
- The checkpointed process cannot use internal timers.
- The checkpointed program must be statically linked.
- `SIGHUP` is used internally to implement checkpointing. Do not use this signal in programs to be checkpointed.

12. Customizing Batch Jobs for LSF

This chapter describes how to customize your batch jobs to take advantage of LSF and LSF Batch features.

Environment Variables

When LSF Batch runs a batch job it sets several environment variables. Batch jobs can use the values of these variables to control how they execute. The environment variables set by LSF Batch are:

LSB_CHKPNT_DIR

This variable is set each time a checkpointed job is submitted. The value of the variable is `chkpntdir/jobId`, a subdirectory of the checkpoint directory that is specified when the job is submitted. The subdirectory is identified by the job ID of the submitted job.

LSB_JOBID

The LSF Batch job ID number.

LSB_JOBFILENAME

The full path name of the batch job file. This is a `/bin/sh` script on UNIX systems or a `.BAT` command script on Windows NT systems that invokes the batch job.

LSB_HOSTS

The list of hosts selected by LSF Batch to run the batch job. If the job is run on a single processor, the value of `LSB_HOSTS` is the name of the execution host. For parallel jobs, the names of all execution hosts are listed separated by spaces. The batch job file is run on the first host in the list.

12 Customizing Batch Jobs for LSF

LSB_QUEUE

The name of the batch queue from which the job was dispatched.

LSB_JOBNAME

The name of the batch job as specified by the `-J job_name` argument to `bsub`. If this argument was not given, the job name is the actual batch command as specified on the `bsub` command line.

LSB_RESTART

If this batch job was submitted with the `-r` option to `bsub`, has run previously, and has been restarted because of a host failure, `LSB_RESTART` is set to the value `Y`. If this is not a restarted job `LSB_RESTART` is not set.

LSB_EXIT_PRE_ABORT

The value of this parameter can be used by a queue or job-level pre-execution command so that the command can exit with this value, if it wants the job be aborted instead of being requeued or executed.

LSB_EXIT_REQUEUE

This variable is a list of exit values defined in the queue's `REQUEUE_EXIT_VALUE` parameter. If this variable is defined, a job will be requeued if it exits with one of these values. This variable is not set if the queue does not have `REQUEUE_EXIT_VALUE` defined.

LSB_JOB_STARTER

This variable is defined if a job starter command is defined for the queue. See '*Queue-Level Job Starters*' on page 129 of the *LSF Batch Administrator's Guide* for more information.

LSB_INTERACTIVE

This variable is set to '`Y`' if the job is an interactive job. An interactive job is submitted using the `-I` option to `bsub`. This variable is not defined if the job is not interactive. See '*Interactive Batch Job Support*' on page 145.

LS_JOBPID

The process ID of the job. This is always a shell script process that runs the actual job.

LS_SUBCWD

This is the directory on the submission host where the job was submitted. By

default LSF Batch assumes a uniform user name and user ID space exists among all the hosts in the cluster, that is, a job submitted by a given user will run under the same user's account on the execution host. For situations where non-uniform user id/user name space exists, account mapping must be used to determine the account used to run a job. See *'User Controlled Account Mapping'* on page 86.

LSB_JOBINDEX

This variable is set only if the job is an element of a job array. The value of the variable is the index of the job into the job array.

Parallel Jobs

Each parallel programming package has different requirements for specifying and communicating with all the hosts used by a parallel job. LSF is not tailored to work with a specific parallel programming package. Instead, LSF provides a generic interface so that any parallel package can be supported by writing shell scripts or wrapper programs. Example shell scripts are provided for running PVM, P4, MPI, and POE programs as parallel batch jobs.

Getting the Host List

The hosts allocated for the parallel job are passed to the batch job in the `LSB_HOSTS` environment variable. Some applications can take this list of hosts directly as a command line parameter. For other applications you may need to process the host list. The following example shows a `/bin/sh` script that processes all the hosts in the host list, including identifying the host where the job script is executing.

```
#!/bin/sh
# Process the list of host names in LSB_HOSTS

for host in $LSB_HOSTS ; do
  handle_host $host
done
```

12 Customizing Batch Jobs for LSF

LSF comes with a few scripts for running parallel jobs under LSF Batch, such as `pvmjob`, `poejob`, `mpijob`, `p4job`, etc. These scripts are installed in the `LSF_BINDIR` as defined in `lsf.conf` file. You can modify these scripts to support more parallel packages.

Starting Parallel Tasks With `lstools`

For simple parallel jobs you can use the `lstools` commands to start parts of the job on other hosts. Because the `lstools` commands handle signals transparently, LSF Batch can suspend and resume all components of your job without additional programming.

The simplest parallel job runs an identical copy of the executable on every host. The `lsgrun` command takes a list of host names and runs the specified task on each host. The `lsgrun -p` option specifies that the task should be run in parallel on each host. The example below submits a job that uses `lsgrun` to run `myjob` on all the selected batch hosts in parallel:

```
% bsub -n 10 'lsgrun -p -m "$LSB_HOSTS" myjob'
Job <3856> is submitted to default queue <normal>.
```

For more complicated jobs, you can write a shell script that runs `lsrun` in the background to start each component.

Using LSF Make to Run Parallel Batch Jobs

For parallel jobs that have a variety of different components to run, you can use LSF Make. Create a makefile that lists all the components of your batch job and then submit the LSF Make command to LSF Batch. The following example shows a `bsub` command and Makefile for a simple parallel job.

```
% bsub -n 4 lsmake -f Parjob.makefile
Job <3858> is submitted to default queue <normal>.
```

```
% cat Parjob.makefile
# Makefile to run example parallel job using lsbatch and LSF Make

all: part1 part2 part3 part4
```

```
part1 part2 part3: myjob data.$@
```

```
part4: myjob2 data.part1 data.part2 data.part3
```

The batch job has four components. The first three components run the `myjob` command on the `data.part1`, `data.part2` and `data.part3` files. The fourth component runs the `myjob2` command on all three data files. There are no dependencies between the components, so LSF Make runs them in parallel.

Submitting PVM Jobs to LSF Batch

PVM is a parallel programming system distributed by Oak Ridge National Laboratories. PVM programs are controlled by a file, the *PVM hosts file*, that contains host names and other information. The `pvmjob` shell script supplied with LSF can be used to run PVM programs as parallel LSF Batch jobs. The `pvmjob` script reads the LSF Batch environment variables, sets up the PVM hosts file and then runs the PVM job. If your PVM job needs special options in the hosts file, you can modify the `pvmjob` script.

For example, if the command line to run your PVM job is:

```
% myjob data1 -o out1
```

the following command submits this job to LSF Batch to run on 10 hosts:

```
% bsub -n 10 pvmjob myjob data1 -o out1
```

Other parallel programming packages can be supported in the same way. The `p4job` shell script runs jobs that use the P4 parallel programming library. Other packages can be handled by creating similar scripts.

Submitting MPI Jobs to LSF Batch

The *Message Passing Interface* (MPI) is a portable library that supports parallel programming. LSF supports MPICH, a joint implementation of MPI by Argonne National Laboratory and Mississippi State University. This version supports both TCP/IP and IBM's *Message Passing Library* (MPL) communication protocols.

LSF provides an `mpijob` shell script that you can use to submit MPI jobs to LSF Batch. The `mpijob` script writes the hosts allocated to the job by the LSF Batch system to a file

12 Customizing Batch Jobs for LSF

and supplies the file as an option to MPICH's `mpirun` command. The syntax of the `mpijob` command is:

```
mpijob option mpirun program arguments
```

Here, *option* is one of the following:

- tcp** Write the LSF Batch hosts to a `PROCGROUP` file, supply the `-p4pg` *procgroupp_file* option to the `mpirun` command, and use the TCP/IP protocol. This is the default.
- mpl** Write the LSF Batch hosts to a `MACHINE` file, supply the `-machinefile` *machine_file* option to the `mpirun` command, and use the MPL on an SP-2 system.

The following examples show how to use `mpijob` to submit MPI jobs to LSF Batch.

To submit a job requesting four hosts and using the default TCP/IP protocol, use:

```
% bsub -n 4 mpijob mpirun myjob
```

Note

Before you can submit a job to a particular pool of IBM SP-2 nodes, an LSF administrator must install the SP-2 ELIM. The SP-2 ELIM provides the pool number and lock status of each node.

To submit the same job to run on four nodes in pool 1 on an IBM SP-2 system using MPL, use:

```
% bsub -n 4 -R "pool == 1" mpijob -mpl mpirun myjob
```

To submit the same job to run on four nodes in pool 1 that are not locked (dedicated to using the High Performance Switch) on an SP-2 system using MPL, use:

```
% bsub -n 4 -q mpiq -R "pool == 1 && lock == 0" mpijob -mpl mpirun myjob
```

Note

Before you can submit a job using the IBM SP-2 High Performance Switch in dedicated mode, an LSF administrator must set up a queue for automatic requeue on job failure. The job queue will automatically requeue a job that failed because an SP-2 node was locked after LSF Batch selected the node but before the job was dispatched.

Submitting POE Jobs to LSF Batch

The *Parallel Operating Environment* (POE) is an execution environment provided by IBM on SP-2 systems to hide the differences between serial and parallel execution.

LSF provides a `poejob` shell script that you can use to submit POE jobs to LSF Batch. The `poejob` script translates the hosts allocated to the job by the LSF Batch system into an appropriate POE host list and sets up environment variables necessary to run the job.

The `poejob` script does not set the `MP_EUILIB` and `MP_EUIDEVICE` environment variables, so you have to do this.

```
% setenv MP_EUILIB us
```

By default, `MP_EUIDEVICE` is `css0`. Or:

```
% setenv MP_EUILIB ip
% setenv MP_EUIDEVICE en0
```

The following are examples of how to submit POE jobs.

To submit a job requesting four SP-2 nodes configured for the `poeq` queue, use:

```
% bsub -n 4 -q poeq poejob myjob
```

By using LSF resource requirements, you can select appropriate nodes for your job.

To submit the same job requesting four SP-2 nodes from pool 2 configured for the `poeq` queue, use:

```
% bsub -n 4 -R "pool == 2" -q poeq poejob myjob
```

To submit the same job requesting four SP-2 nodes from pool 2 with at least 20 megabytes of swap space, use:

```
% bsub -n 4 -R "(pool == 2) && (swap > 20)" -q poeq poejob myjob
```

12 Customizing Batch Jobs for LSF

To submit the same job requesting four SP-2 nodes from pool 2 that are not locked (dedicated to using the High Performance Switch), use:

```
% bsub -n 4 -R "(pool == 2) && (lock == 0)" -q poeq poejob myjob
```

Using a Job Starter for Parallel Jobs

The above examples use a script to run parallel jobs under LSF Batch. Alternatively, your LSF administrator could configure the script into your queue as a job starter. With a job starter configured at the queue, you can submit the above parallel jobs without having to type the script name. See *'Queue-Level Job Starters' on page 129 of the LSF Batch Administrator's Guide* for more information about job starters.

To see if your queue already has a job starter defined, run `bqueues -l` command.

13. Using LSF MultiCluster

What is LSF MultiCluster?

Within a company or organization, each division, department, or site may have a separately managed LSF cluster. Many organizations have realized that it is desirable to allow their multitude of clusters to cooperate to reap the benefits of global load sharing:

- You can access a diverse collection of computing resources and get better performance as well as computing capabilities. Many machines that would otherwise be idle can be used to process jobs. Multiple machines can be used to process a single parallel job. All these lead to increased user productivity.
- The demands for computing resources fluctuate widely across departments and over time. Partitioning the resources of an organization along user and departmental boundaries forces each department to plan for computing resources according to its maximum demand. Load sharing makes it possible for an organization to plan computing resources globally based on total demand. Resources can be added anywhere and made available to the entire organization. Global policies for load sharing can be implemented. With efficient resource sharing, the organization can realize increased computer usage in an economical manner.

LSF MultiCluster enables a large organization to form multiple cooperating clusters of computers so that load sharing happens not only within the clusters but also among them. It enables load sharing across large numbers of hosts, allows resource ownership and autonomy to be enforced, non-shared user accounts and file systems to be supported, and communication limitations among the clusters to be taken into consideration in job scheduling.

Getting Remote Cluster Information

The commands `lshosts`, `lsload`, and `lsmon` can accept a cluster name to allow you to view the remote cluster. A list of clusters and associated information can be viewed with the `lsclusters` command.

% `lsclusters`

CLUSTER_NAME	STATUS	MASTER_HOST	ADMIN	HOSTS	SERVERS
clus1	ok	hostC	user1	3	3
clus2	ok	hostA	user1	3	3

% `lshosts`

HOST_NAME	type	model	cpuf	ncpus	maxmem	maxswp	server	RESOURCES
hostA	NTX86	PENT200	10.0	-	-	-	Yes	(NT)
hostF	HPPA	HP735	14.0	1	58M	94M	Yes	(hpux cserver)
hostB	SUN41	SPARCSLC	3.0	1	15M	29M	Yes	(sparc bsd)
hostD	HPPA	HP735	14.0	1	463M	812M	Yes	(hpux cserver)
hostE	SGI	R10K	16.0	16	896M	1692M	Yes	(irix cserver)
hostC	SUNSOL	SunSparc	12.0	1	56M	75M	Yes	(solaris cserver)

% `lshosts clus1`

HOST_NAME	type	model	cpuf	ncpus	maxmem	maxswp	server	RESOURCES
hostD	HPPA	HP735	14.0	1	463M	812M	Yes	(hpux cserver)
hostE	SGI	R10K	16.0	16	896M	1692M	Yes	(irix cserver)
hostC	SUNSOL	SunSparc	12.0	1	56M	75M	Yes	(solaris cserver)

% `lshosts clus2`

HOST_NAME	type	model	cpuf	ncpus	maxmem	maxswp	server	RESOURCES
hostA	NTX86	PENT200	10.0	-	-	-	Yes	(NT)
hostF	HPPA	HP735	14.0	1	58M	94M	Yes	(hpux cserver)
hostB	SUN41	SPARCSLC	3.0	1	15M	29M	Yes	(sparc bsd)

% `lsload clus1 clus2`

HOST_NAME	status	r15s	r1m	r15m	ut	pg	ls	it	ttmp	swp	mem
hostD	ok	0.2	0.3	0.4	19%	6.0	6	3	146M	319M	252M
hostC	ok	0.1	0.0	0.1	1%	0.0	3	43	63M	44M	27M
hostA	ok	0.3	0.3	0.4	35%	0.0	3	1	40M	42M	13M
hostB	busy	*1.3	1.1	0.7	68%	*57.5	2	4	18M	20M	8M
hostE	lockU	1.2	2.2	2.6	30%	5.2	35	0	10M	693M	399M
hostF	unavail										

Running Batch Jobs across Clusters

A queue may be configured to send LSF Batch jobs to a queue in a remote cluster (see *'LSF Batch Configuration' on page 148 of the LSF Batch Administrator's Guide*). When you submit a job to that local queue it will automatically get sent to the remote cluster:

The `bclusters` command displays a list of local queues together with their relationship with queues in remote clusters.

% **bclusters**

LOCAL_QUEUE	JOB_FLOW	REMOTE	CLUSTER	STATUS
testmc	send	testmc	clus2	ok
testmc	recv	-	clus2	ok

The meanings of displayed fields are:

LOCAL_QUEUE

The name of the local queue that either receive jobs from queues in remote clusters, or forward jobs to queues in remote clusters.

JOB_FLOW

The value can be either `send` or `recv`. If the value is `send`, then this line describes a job flow from the local queue to a queue in a remote cluster. If the value is `recv`, then this line describes a job flow from a remote cluster to the local queue.

REMOTE

Queue name of a remote cluster that the local queue can send jobs to. This field is always "-" if `JOB_FLOW` field is "`recv`".

CLUSTER

Remote cluster name.

STATUS

Connection status between the local queue and remote queue. If `JOB_FLOW` field is `send`, then the possible values for `STATUS` field are "`ok`", "`reject`", and "`disc`", otherwise the possible status are "`ok`" and "`disc`". When status is "`ok`", it indicates that both queues agree on the job flow. When status is "`disc`", it means communications between the local and remote cluster has

13 Using LSF MultiCluster

not been established yet. This may either be because no jobs need to be forwarded to the remote cluster yet, or the `mbatchd`'s of the two clusters have not been able to get in touch with each other. The `STATUS` is `reject` if `send` is the job flow and the queue in the remote cluster is not configured to receive jobs from the local queue.

In the above example, local queue `testmc` can forward jobs in the local cluster to `testmc` queue of remote cluster `clus2` and vice versa.

If there is no queue in your cluster that is configured for remote clusters, you will see the following:

```
% bclusters
```

```
No local queue sending/receiving jobs from remote clusters
```

Use the `-m` option with a cluster name to the `bqueues` command to display the queues in the remote cluster.

```
% bqueues -m clus2
```

QUEUE_NAME	PRIOR	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
fair	3300	Open:Active	5	-	-	-	1	1	0	0
interactive	1055	Open:Active	-	-	-	-	1	0	1	0
testmc	55	Open:Active	-	-	-	-	5	2	2	1
priority	43	Open:Active	-	-	-	-	0	0	0	0

Submit your job with the `bsub` command to the queue that is sending jobs to the remote cluster.

```
% bsub -q testmc -J mcjob myjob
```

```
Job <101> is submitted to queue <testmc>.
```

The `bjobs` command will display the cluster name in the `FROM_HOST` and `EXEC_HOST` fields. The format of these fields is 'host@cluster' indicating which cluster the job originated from or was forwarded to. To query the jobs running in another cluster, use the `-m` option and specify a cluster name.

```
% bjobs
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
101	user7	RUN	testmc	hostC	hostA@clus2	mcjob	Oct 19 19:41

```
% bjobs -m clus2
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
522	user7	RUN	testmc	hostC@clus2	hostA	mcjob	Oct 19 23:09

Note that the submission time shown from the remote cluster is the time when the job was forwarded to that cluster.

To view the hosts of another cluster you can use a cluster name in place of a host name as the argument to the `bhosts` command.

```
% bhosts clus2
```

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
hostA	ok	-	10	1	1	0	0	0
hostB	ok	-	10	2	1	0	0	1
hostF	unavail	-	3	1	1	0	0	0

Run `bhist` command to see the history of your job, including information about job forwarding to another cluster.

```
% bhist -l 101
```

```
Job Id <101>, Job Name <mcjob>, User <user7>, Project <default>, Command
    <myjob>
Sat Oct 19 19:41:14: Submitted from host <hostC> to Queue <testmc>,CWD <$HOME>
Sat Oct 19 21:18:40: Parameters are modified to:Project <test>,Queue <testmc>,
    Job Name <mcjob>;
Mon Oct 19 23:09:26: Forwarded job to cluster clus2;
Mon Oct 19 23:09:26: Dispatched to <hostA>;
Mon Oct 19 23:09:40: Running with execution home </home/user7>, Execution CWD <
    /home/user7>, Execution Pid <4873>;
Mon Oct 20 07:02:53: Done successfully. The CPU time used is 12981.4 seconds;
```

```
Summary of time in seconds spent in various states by Wed Oct 20 07:02:53 1997
```

PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
5846	0	28399	0	0	0	34245

Running Interactive Jobs on Remote Clusters

The `lsrun` command allows you to specify a cluster name instead of a host name. When a cluster name is specified, a host is selected from the cluster. For example:

```
% lsrun -m clus2 -R type==any hostname  
hostA
```

The `-m` option to the `lslogin` command can be used to specify a cluster name. This allows you to login to the best host in a remote cluster.

```
% lslogin -v -m clus2  
<<Remote login to hostF >>
```

The multicluster environment can be configured so that one cluster accepts interactive jobs from the other cluster, but not vice versa. See ‘*Running Interactive Jobs on Remote Clusters*’ on page 152 of the *LSF Batch Administrator’s Guide*. If the remote cluster will not accept jobs from your cluster, you will get an error:

```
% lsrun -m clus2 -R type==any hostname  
ls_placeofhosts: Not enough host(s) currently eligible
```

User-Level Account Mapping Between Clusters

By default, LSF assumes a uniform user name space within a cluster and between clusters. It is not uncommon for an organization to fail to satisfy this assumption. Support for non-uniform user name spaces between clusters is provided for the execution of batch jobs. The `.lsfhosts` file used to support account mapping can be used to specifying cluster names in place of host names.

For example, you have accounts on two clusters, *clus1* and *clus2*. In *clus1*, your user name is ‘*user1*’ and in *clus2* your user name is ‘*ruser_1*’. To run your jobs in either cluster under the appropriate user name, you should setup your `.lsfhosts` file as follows:

On machines in cluster *clus1*:

```
% cat ~user1/.lsfhosts
clus2 ruser_1
```

On machines in cluster *clus2*:

```
% cat ~ruser_1/.lsfhosts
clus1 user1
```

For another example, you have the account ‘*user1*’ on cluster *clus1* and you want to use the ‘*lsfguest*’ account when sending jobs to be run on cluster *clus2*. The *.lsfhosts* files should be setup as follows:

On machines in cluster *clus1*:

```
% cat ~user1/.lsfhosts
clus2 lsfguest send
```

On machines in cluster *clus2*:

```
% cat ~lsfguest/.lsfhosts
clus1 user1 recv
```

The other features of the *.lsfhosts* file also work in the multicluster environment. See ‘*User Controlled Account Mapping*’ on page 86 for further details. Also see ‘*Account Mapping Between Clusters*’ on page 155 of the *LSF Batch Administrator’s Guide*.

14. Interoperation with NQS

The Network Queuing System (NQS) is a UNIX batch queuing facility that allows users to queue batch jobs to individual UNIX hosts from remote systems. Many users have been using NQS for years.

This chapter describes how LSF works with existing NQS systems. If you are not going to use LSF to interoperate with NQS, you do not need to read this chapter.

For user sites who have been using NQS for years, LSF provides a set of NQS compatible commands for them to submit jobs to LSF using the NQS command syntax. Examples of NQS compatibility commands in LSF include `qsub`, `qstat`, and `qdel`.

While it is desirable to run LSF on all hosts for transparent resource sharing, this is not always possible. Some of the computing resources may be under separate administrative control, or LSF may not currently be available for some of the hosts.

An example of this are sites that use Cray supercomputers. The supercomputer is often not under the control of the workstation system administrators. Users on the workstation cluster still want to run jobs on the Cray supercomputer. LSF allows users to submit and control jobs on the Cray system using the same LSF interface as they use for jobs on the local cluster.

LSF queues can be configured to forward jobs to remote NQS queues. Users can submit jobs, send signals to jobs, check status of jobs, and delete jobs that are forwarded to the remote NQS. Although running on an NQS server outside the LSF cluster, jobs are still managed by LSF Batch in almost the same way as jobs running inside the LSF cluster.

Choosing an LSF Batch Queue

To submit jobs to hosts where NQS is running, you first need to find out which LSF Batch queues are configured to forward jobs to NQS. The `bqueues -l` command lists detailed information about all LSF Batch queues. Queues that have the 'NQS DESTINATION QUEUES' parameter defined will forward jobs to remote NQS hosts. Below is an example of the output from the `bqueues` command that describes such a queue:

```
% bqueues -l cray
QUEUE: cray
-- For jobs to be sent to the Cray supercomputer.

PARAMETERS/STATISTICS
PRIO NICE STATUS      MAX   JL/U  JL/P  NJOBS  PEND  RUN  SSUSP  USUSP  RSV
30   15   Open:Active 5     -    -    1      0     1    0     0    0

SCHEDULING PARAMETERS
          r15s  r1m  r15m  ut    pg    io  ls   it   tmp   swp  mem
loadSched -    -    -    -    -    -    -   -   -    -    -
loadStop  -    -    -    -    -    -    -   -   -    -    -

USERS:  all users
NQS DESTINATION QUEUES: nqs_queue@crayhost.company.com
```

Note that 'nqs_queue' in the above output is the name of the NQS queue on the specified host.

Submitting a Job from LSF to NQS

Submitting a job to run on an NQS host is the same as submitting an ordinary LSF job, except that only those options that reflect common functionality of both LSF and NQS can be used. This is because some NQS options do not make sense in the LSF context, and many LSF options are not supported by NQS. Options must be specified as LSF options; they are automatically translated when the job is forwarded to NQS. See the

LSF `bsub(1)` manual page and the NQS `qsub(1)` manual page for more information on the options supported by LSF and NQS.

Controlling Jobs Running on NQS

Job information from NQS is translated by LSF and reported by LSF Batch commands. Any signals supported by both LSF and NQS may be sent to a specified job.

Forwarding of Output Files

The `stdout` and `stderr` output of the job is always transferred from the NQS host back to the LSF cluster. If the `bsub -o` or `-e` options are not specified, the output of the job is mailed to the user. If either of the `-o` or `-e` options are specified, the output received from the NQS server is stored in the specified files.

A. Customizing xlsbatch Menu Items

You can customize the menu items of `xlsbatch` by specifying `customizedMenu` resource in the X resource file. With this feature, you can:

- remove a pull-down menu from the menu bar
- remove a menu item from a pull-down menu
- remove a sub-pull-down menu from a pull-down menu
- remove a menu item from a sub-pull-down menu
- add a new item in the menu bar
- add a new item in a pull-down menu with an executable (the executable is run when the menu item is chosen)
- add a new item in a sub-pull-down menu with an executable
- replace an item's command in the pull-down menu by a new executable
- replace an item's command in the sub-pull-down menu by a new executable

The format of the `customizedMenu` resource is:

```
xlsbatch*customizedMenu: 'A1 B1 C1 D1 A2 B2 C2 B2 ... An Bn Cn Dn'
```

where the resource value is a character string enclosed in double quotes, whose components are separated by spaces.

A Customizing xlsbatch Menu Items

From the beginning of the string, each four consecutive components constitute a group that customizes one menu item. Each component is either a character string or several character strings encompassed by parentheses.

The following define the valid syntax for a menu item group:

- Remove a pull-down menu from the menu bar:
Ai: the pull-down menu name to be removed from the menu bar
Bi: 0 (zero)
Ci: 0 (zero)
Di: 0 (zero)

For example, remove the 'Calendar' pull-down menu from the menu bar:

```
xlsbatch*customizedMenu:"Calendar 0 0 0"
```

- Remove a menu item from a pull-down menu:
Ai: the pull-down menu name where some item will be removed
Bi: the menu item name to be removed
Ci: 0 (zero)
Di: 0 (zero)

For example, remove the 'Restart...' menu item from the 'Job' pull-down menu:

```
xlsbatch*customizedMenu:"Job Restart... 0 0"
```

- Remove a sub-pull-down menu from a pull-down menu:
Ai: the pull-down menu where some sub-pull-down will be removed
Bi: the sub-pull-down menu name to be removed
Ci: 0 (zero)
Di: 0 (zero)

For example, remove the 'View' sub-pull-down menu from the 'Job' pull-down menu:

```
xlsbatch*customizedMenu:"Job View 0 0"
```

- Remove a menu item from a sub-pull-down menu:
Ai: the pull-down menu where the sub-pull-down menu resides

Bi: the sub-pull-down menu name where some item will be removed

Ci: the menu item name to be removed

Di: 0 (zero)

For example, remove the 'Chkpnt . . .' menu item from the 'Manipulate' sub-pull-down menu of the 'Job' pull-down menu:

```
xlsbatch*customizedMenu:"Job Manipulate Chkpnt... 0"
```

- Add a new item in the menu bar:

Ai: new menu name to be added

Bi: 0 (zero)

Ci: 0 (zero)

Di: same as *Ai*

For example, add the 'Others' menu item in the menu bar:

```
xlsbatch*customizedMenu:"Others 0 0 Others"
```

- Add a new item in a pull-down menu with an executable:

Ai: the pull-down menu name where the new item will be added

Bi: new menu item name to be added

Ci: 0 (zero)

Di: executable name which will be executed when this item is chosen

For example, add the 'xlsadmin' menu item in the 'File' pull-down menu which runs xlsadmin with gray background:

```
xlsbatch*customizedMenu:"File xlsadmin 0 (xlsadmin -bg gray)"
```

- Add a new item in a sub-pull-down menu with an executable:

Ai: the pull-down menu name where the sub-pull-down menu resides

Bi: the sub-pull-down menu name where the new item will be added

Ci: new menu item name to be added

Di: executable name which will be executed when this item is chosen

For example, add the 'My Job View' menu item in the 'View' sub-pull-down menu of the 'Job' pull-down menu, which runs the myView executable:

```
xlsbatch*customizedMenu:"Job View (My Job View) myView"
```

A Customizing xlsbatch Menu Items

- Replace an item's command in the pull-down menu by a new executable:
Ai: the pull-down menu name where some item's command will be replaced
Bi: the menu item name whose command will be replaced
Ci: 0
Di: executable name which will be executed when the item specified in *Bi* is chosen

For example, replace the 'Submit...' menu item's command in the 'Job' pull-down menu by a new `myBsub` executable:

```
xlsbatch*customizedMenu:"Job Submit... 0 myBsub"
```

- Replace an item's command in the sub-pull-down menu by a new executable:
Ai: the pull-down menu name where the sub-pull-down menu resides
Bi: the sub-pull-down menu name where some item's command will be replaced
Ci: the menu item name whose command will be replaced
Di: executable name which will be executed when the item specified in *Ci* is chosen

For example, replace the 'Chkpnt...' menu item's command in the 'Manipulate' sub-pull-down menu by a new `myChkpnt` executable:

```
xlsbatch*customizedMenu:"Job Manipulate Chkpnt... myChkpnt"
```

`xlsbatch` puts the jobIDs of those selected in the job list of the Main Window into a group of UNIX environment variables:

```
JOBID0 JOBID1 ... JOBIDk
```

Starting with `JOBID0`, each variable contains at most 200 job IDs separated by spaces, the variable group ends with a null variable.

B. Frame Arrays

Extending job array support, the frame array enables a subset of jobs in an array to be specified as a manageable unit. It can be associated with its own attributes such as resource requirements and job processing parameters. A frame array can be scheduled, modified, and controlled just like a single job.

In this appendix, we assume that you are already familiar with job arrays in LSF.

Overview

The frame array provides a structure that allows all the frames that make up a scene to be submitted using a single command. The frame array also provides the ability to control and manipulate individual frames, frame sub-sequences, and the whole scene (frame array). It groups frames (elements in a frame array) into user specified manageable units (chunks) that are submitted to the LSF Batch system as individual jobs. All jobs derived from a frame array share the same job ID and submission parameters.

Frames in a frame array are referenced using an array index, which supports negative indices. The dimension and structure of a frame array are defined when the array is submitted. Frames are scheduled to run independently of each other using the various policies that apply to the submitting user.

Frame arrays are especially useful for the Computer Graphics Industry (CGI) where the process of rendering a sequence of individual frames to build a scene is an everyday occurrence. The frames that make up a scene represent independent entities. They can be processed separately, therefore using the frame array is a natural approach to improving rendering throughput.

B Frame Arrays

The default size of a frame array is 1000 jobs, the maximum size is 2046 jobs. The `MAX_JOB_ARRAY_SIZE` parameter specified in the `lsb.params` file sets the size of a frame array. The frame array and the individual frames are modified and controlled using the following new commands:

- `gbsub`
submits a frame array to the LSF Batch system
- `gbjobs`
displays the frame status for frame array jobs
- `gbstop`
suspends a frame array or individual frames
- `gbresume`
resumes a suspended frame array or individual frames
- `gbkill`
sends a signal to a frame array or individual frames

Distribution

The files that allow LSF to use frame arrays are available from Platform. Installation instructions are included.

Frame Array Concepts

frame

The basic element that makes up a frame array, hence the name. As the name implies, a frame is one picture of a series on a length of film. A number of frames make up a scene.

job

In the context of frame arrays, a job is the smallest object of control within a frame array. Jobs are defined by the chunk parameter.

chunk

The number of frames that are to be grouped together to form a job.

frame array

The structure used to define the dimension (number of frames) and structure (chunks) of the frames that make up the scene. The frame array is the structure that is submitted to the LSF Batch system.

Submitting Frame Arrays

A frame array is created at the time of submission. The `gbsub` command inherits all the options of the `bsub` command and extends the `-J` option to specify the elements of the array. Each element of the array corresponds to a single frame and is identified by an index which must be a unique integer. The index values in an array do not have to be consecutive, and a combination of individual index values and index ranges are used to define a frame array.

Syntax

```
% gbsub -J "frameArrayName[indexList, . . .]" command
```

Note

The frame array specification must be enclosed in double quotes.

The square brackets, [], around indexList must be entered exactly as shown.

Note: The frame array syntax breaks the convention of using square brackets to indicate optional items

frameArrayName

Specifies a user defined string used to name the frame array. Any combination of the following characters make up a valid frameArrayName:

B Frame Arrays

a-z | A-Z | 0-9 | . | - | _

indexList

Specifies the dimension, structure, and indices of the frame array in the following format:

```
indexList = start [- end [x step [: chunk]]]
```

start

A unique integer specifying the start of a range of frame array indices. If a start value is specified without an end value, start specifies an individual job of the frame array.

end

A unique integer specifying the end of a range of frame array indices. The start-end construct specifies a range of frames.

step

An integer specifying the value to increment the index values for the respective range of frames. If omitted, the default value is 1.

chunk

A single integer specifying the number of frames to group (chunk) into a job. If omitted, default value is 1.

A job is the smallest object of control within the LSF Batch system, when chunk size is larger than 1 the individual frames that make up the job are controlled as a single job. When the number of frames cannot be evenly divided by chunk size, the last job (chunk) will contain fewer frames.

The following formulas help to explain the relationship between frames, steps, and chunks.

Note

In these examples, if the number of frames or jobs turns out to be a fraction, it should always be rounded up to the next integer. For example, 9/3 would be 3, but 10/3 would be 4.

$$\text{Number of Frames} = \frac{(\text{end} - \text{start}) + 1}{\text{step}}$$

$$\text{Number of Jobs} = \frac{\text{Number of Frames}}{\text{chunk}}$$

Examples

- [1] specifies 1 frame with the index of 1 submitted as 1 job.
- [1, 2, 3, 4, 5] specifies 5 frames with indices 1 - 5 submitted as 5 jobs .
- [1-5] specifies 5 frames with indices 1 - 5 submitted as 5 jobs. Step and chunk values are default (i.e., 1). Index values are determined by starting at 1 and adding 1, not incrementing past the end value.
- [1-10x2] specifies 5 frames with indices 1, 3, 5, 7, and 9 submitted as 5 jobs. Step value is 2, chunk value is default (i.e., 1). Index values are determined by starting at 1 and adding 2, not incrementing past the end value.
- [1-10x2:3] specifies a frame array of 5 frames with indices 1, 3, 5, 7, and 9 submitted as 2 jobs. Step value is 2 and chunk value is 3. Index values are determined by starting at 1 and adding 2. Jobs are determined by putting frames in groups of 3. When chunk size does not evenly divide frame size the remainder frames (i.e., < chunk) make up the last job.
- [-99-0, 1-100] specifies a frame array with indices -99 to 0 and 1 to 100 submitted as 199 jobs. Step and chunk values are default (i.e., 1).
- [-99-0x10:5, 1-100x5:10] specifies the following frame array:
 - Step value 10, chunk size 5, with indices -99, -89, -79, -69, -59, -49, -39, -29, -19, and -9 submitted as 2 jobs:
 - -99, -89, -79, -69, -59
 - -49, -39, -29, -19, -9
 - Step value 5, chunk size 10, with indices 1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81, 86, 91, and 96 submitted as 2 jobs:
 - 1, 6, 11, 16, 21, 26, 31, 36, 41, 46
 - 51, 56, 61, 66, 71, 76, 81, 86, 91, 96

Tracking Frame Arrays

The `gbjobs` command is used to track frame arrays. It inherits all the options of `bjobs` command with the addition of the `-Jn "frameArrayName"` and `-Js "frameArrayName"` options. The `-Jn` option displays frame jobs (chunks) arranged by the frame index and job state. The `-Js` option displays the status of all jobs in a frame array.

Examples

A frame array is submitted:

```
% gbsub -J "myFrame[-100--50x10:3,-40--20x10:3,-10,0,100-10x-10]" command
```

The command `gbjobs -Jn "myFrame"` produces output that looks like:

JOBGID	USER	FRAMES	STAT
101	user1	myFrame[-100--50x10:3]	RUN
101	user1	myFrame[-40--20x10:3]	PEND
101	user1	mtFrame[-10]	PEND
101	user1	myFrame[0]	PEND
101	user1	myFrame[100-10x-10]	PSUSP

The command `gbjobs -Js "myFrame"` produces output that looks like:

JOBGID	FRAMES	PEND	RUN	DONE	EXIT	SUSP	OTH
101	myFrame[-100--20x10:3,-10,0,100-10x-10]	3	2				10

Controlling Frame Arrays

The `gbstop`, `gbresume`, and `gbkill` commands are used to control frame arrays. These commands inherit all the options of the `bstop`, `bresume` and `bkill` commands respectively, except for `-R`, `-d`, and `-a`, which are not supported. For frame arrays the `-J "frameArrayName[indexList]"` option operates at the job level, which is

defined by the chunk size at submission time. These commands echo the frames on which they will be operating.

Syntax

```
gbstop [options] -J "frameArrayName[indexList, . . .]" command
gbresume [options] -J "frameArrayName[indexList, . . .]" command
gbkill [options] -J "frameArrayName[indexList, . . .]" command
-J "frameArrayName[indexList]"
```

The syntax for `frameArrayName[indexList]` is the same used with the `gbsub` command. The asterisk, `*`, wildcard is supported for `frameArrayName`. Care must be used with the wildcard, as it will cause the command to operate all arrays whose names match the pattern. Chunk size is ignored for these commands. If only `frameArrayName` is specified (i.e., without an `indexList`) the command will operate on all jobs whose name matches `frameArrayName`.

If a mistake is made when issuing a control (i.e., indexing a frame that does not exist) the following error will be reported:

```
lsb_openjobinfo: No matching job found
```

Examples

A frame array is submitted using the following specification:

```
% gbsub -J "myFrameArray[1-10x2:3]" command
```

`[1-10x2:3]` specifies a frame array of 5 frames with indices 1, 3, 5, 7, and 9 submitted as 2 jobs. Step value is 2 and chunk value is 3. Index values are determined by starting at 1 and adding 2. Jobs are determined by putting frames in groups of 3. When chunk size does not evenly divide frame size the remainder frames (i.e., `< chunk`) make up the last job.

```
gbkill -J "myFrameArray[10]" command
```

specifies the frame at index 10 is to be killed. If `myFrameArray[10]` does not exist, the following error message is issued in response: `lsb_openjobinfo: No matching job found.`

B Frame Arrays

`gbstop -J "myFrameArray[1]" command`

specifies the job at index 1 is to be stopped. Since that frame belongs to a job (chunk) whose size is larger than 1, all frames in the job (1 and 3) are stopped.

`gbresume -J "myFrameArray[3]" command`

specifies the frame at index 3 is to be started. Since that frame belongs to a job (chunk) whose size is larger than 1, all frames in the job (1 and 3) are started.

C. Using LSF with Alias Renderer

LSF is integrated with Alias | Wavefront's product Alias Renderer. This allows frame array jobs to take advantage of the checkpoint and migration features provided by LSF Batch, and eliminates the need to reprocess completed frames when an interrupted job is migrated or restarted.

In this appendix, we assume that you are already familiar with frame arrays and job starters in LSF.

Overview

When processing a set of frames, it is not convenient to submit each frame as a separate job. In LSF, a frame array is divided into a specified number of related jobs, and they can all be controlled by a single frame array command. Each job may contain several frames.

This is still not the most efficient way to process the data. For example, if a job consisting of five frames is interrupted while processing the fourth frame, the entire job is restarted elsewhere. Ideally, LSF would be aware that three frames have already been completed, and only the remaining fourth and fifth frames would be processed when a job is restarted. This is possible using the Alias | Wavefront software and special LSF commands.

LSF provides wrappers used to append frame parameters (start, end, and step) to the normal Alias Renderer commands. The Alias | Wavefront frame rendering tools call a specified callback function to export data when each frame is done. LSF provides that function, and updates a file that keeps track of the status of each frame in a job. LSF also provides a queue-level job starter that reads information in this status file before starting a job, so if a job is restarted, only the incomplete frames are submitted for processing.

This feature is called checkpointing. You can use a special LSF command to view the status of frames in checkpointed frame arrays.

Distribution

The distributed rendering control software that you can use with Alias | Wavefront's software is available from Platform. Installation instructions are included.

Installing the queue-level job starter

For frame array checkpointing to work, the frame array must be submitted to a queue that uses the special job starter provided by LSF. Include the following line in the queue definition (in the `lsb.queues` file):

```
JOB_STARTER = gbstarter
```

Submitting Checkpointed Frame Arrays

Submit the frame array as usual, but include the parameters required for checkpointing. The syntax for the `gbsub` command is:

```
gbsub [-k chkpntDir]  
      [-q gbstarter_queue]  
      -J "frameArrayName [IndexList,...]"  
      [any other regular options to the bsub command]  
      gbrenderer | gbraytracer | gbpowercaster | gbpowertracer  
      [-f gbcallback]  
      [any other regular options to the Renderer command]
```

The frame array checkpointing feature requires all the following parameters:

`-k chkpntDir`

Regular option to `gbsub` command, specifies the name of the checkpoint directory that will contain the status files which keep track of completed frames in each job. This directory must already exist; LSF will not create it for you.

`-q gbstarter_queue`

Regular option to `gbsub` command, specifies the name of a queue that uses `gbstarter` as the job starter.

`-J "frameArrayName [IndexList,...]"`

Regular parameter of the `gbsub` command, specifies the name and structure of the frame array.

gb* Renderer commands

Special commands replace the regular parameter of the `gbsub` command. These wrappers must be used to replace the Alias Renderer commands `renderer`, `raytracer`, `powercaster`, and `powertracer`. The wrappers accept all the command options accepted by the native commands, and can be used in scripts just like the native commands.

`-f gbcallback`

Special option to the `gb*` Renderer command, specifies the LSF function `gbcallback` that is called by the Alias Renderer application each time a frame is finished. This function automatically updates the status files in the specified checkpoint directory.

Example

You already have a directory called `MyDir` and the LSF job starter `gbstarter` is configured on a queue called `MyQueue`. If you would normally submit your frame array using the Alias Renderer `powertracer` command, type the following command to enable checkpointing:

```
gbsub -k MyDir -q MyQueue -J "MyArray [1-10x2:3]" gbpowertracer -f gbcallback
```

Tracking Checkpointed Frame Arrays

The regular `gbjobs` command is used with a special switch, `-Jf`, to show the status of frames in checkpointed frame arrays. If you have not submitted any checkpointed frame arrays, an error message will appear.

Example

The command

```
gbjobs -u all -Jf "myFrame"
```

produces output that looks like:

JOBID	USER	FRAMES	EXEC HOST	FRAMES DONE	FRAMES TO GO
5	usr1	myjob1[1-10]	HostA	1-3	4-10
6	usr1	myjob2[17-20]	HostB	17-20	none
9	usr2	seq1[-1-12]	HostC	none	-1-12
15	usr2	seq6[100-200]	HostD	100-138	139-200
25	usr3	mov[500-1000]	HostE	500-813	814-1000

D. Using LSF with FLUENT

LSF is integrated with products from Fluent Inc., allowing FLUENT jobs to take advantage of the checkpoint and migration features provided by LSF Batch. This increases the efficiency of the software and means the data is processed faster.

In this appendix, we assume that you are already familiar with using FLUENT software and checkpointing jobs in LSF.

Overview

For checkpointing jobs, LSF uses two executable files called `echkpnt` and `erestart`. LSF provides special versions of `echkpnt` and `erestart` that will allow checkpointing with FLUENT software.

When you submit a checkpointing job, you have to specify a checkpoint directory. Before the job starts running, LSF sets the environment variable `LSB_CHKPNT_DIR`. The value of `LSB_CHKPNT_DIR` is a subdirectory of the checkpoint directory specified in the command line. This subdirectory is identified by the job ID and will only contain files relating to the submitted job.

When you checkpoint the FLUENT job, LSF creates a checkpoint trigger file (`.check`) in the job subdirectory, which will cause the FLUENT software to checkpoint and continue running. A special option is used to create a different trigger file (`.exit`) which will cause the FLUENT software to checkpoint and exit the job.

The FLUENT software uses the `LSB_CHKPNT_DIR` environment variable to determine the location of checkpoint trigger files. It checks the job subdirectory periodically while running the job. The FLUENT software does not do any checkpointing unless it finds the LSF trigger file in the job subdirectory. The FLUENT software removes the trigger file after checkpointing the job.

If a job is restarted, LSF will attempt to restart the job with "-r" option appended to the original FLUENT command. FLUENT software will use the checkpointed data and case files to restart the process from that checkpoint point, rather than repeating the entire process.

Each time a job is restarted, it is assigned a new job ID, and a new job subdirectory is created in the checkpoint directory. Files in the checkpoint directory are never deleted by LSF, but you may choose to remove old files once the FLUENT job is finished and the job history is no longer required.

Distribution

The files that you can use with FLUENT software are available from Platform. Installation instructions are included.

Configuring the Checkpointing Executable Files

LSF provides special versions of `echkpnt` and `erestart` that will allow checkpointing with FLUENT software. You must make sure LSF uses these files instead of the standard versions. There are two ways to do this:

- Put them in the normal location, so you overwrite the standard LSF files with the special FLUENT versions.
- Leave the standard LSF files in the default location and install the FLUENT versions in a different directory. Then modify the `LSF_ECHKPNTDIR` environment variable to point to the FLUENT versions.

Note

The `LSF_ECHKPNTDIR` environment variable, defined in the `lsf.conf` file, specifies the location of the `echkpnt` and `erestart` files that LSF will use. If this variable is not defined, LSF uses the files in the default location, identified by the environment variable `LSF_SERVERDIR`.

Submitting the FLUENT Job

Submit the job as usual, but include the parameters required for checkpointing. The syntax for the `bsub` command is:

```
bsub [-k chkpntDir]
    [any other regular options to the bsub command]
    FLUENT command
    [any other regular options to the FLUENT command]
    - lsf
```

The checkpointing feature for FLUENT jobs requires all the following parameters:

- k *chkpntDir*
Regular option to `bsub` command, specifies the name of the checkpoint directory.
- FLUENT command*
The regular command used with FLUENT software.
- lsf
Special option to the FLUENT command. Specifies that the FLUENT software is running under LSF, and causes the FLUENT software to check for trigger files in the checkpoint directory if the environment variable `LSB_CHKPNT_DIR` is set.

Note

This option to the FLUENT command should be documented with the FLUENT software. At the time of printing, the option was `-lsf`, but this may change.

Checkpointing the FLUENT job

Checkpoint the FLUENT job manually. The syntax for the `bchkpnt` command is:

```
bchkpnt [regular options to bchkpnt] [-k] [jobId]
```

The following parameters are used with FLUENT:

`-k`

Regular option to `bchkpnt` command, specifies checkpoint and exit. The job will be killed immediately after being checkpointed. When the job is restarted, it doesn't have to repeat any operations.

`jobId`

Job ID of the FLUENT job, should be used to specify which job to checkpoint.

Restarting the FLUENT job

Restart the FLUENT job as usual. The syntax for the `brestart` command is:

```
brestart [regular options to brestart] chkpntDir [jobId]
```

The following parameters are used with FLUENT:

chkpntDir

Specifies the checkpoint directory, where the job subdirectory is located.

jobId

Job ID of the FLUENT job, specifies which job to restart. At this point, the restarted job is assigned a new job ID, and the new job ID starts being used for checkpointing. The job ID changes each time the job is restarted.

Index

A

access permissions 139
address (Platform) xv
administrator, *see* LSF administrator
AFS (Andrew File System) 139
Alias | Wavefront software 211
aliases for resource names 47
API (Application Programming
 Interface) 7
 LSBLIB (LSF Batch LIBrary) 11
 LSLIB (Load Sharing LIBrary) 6
application programming 6
arrays
 frame arrays 203
 job arrays 106
at sign (@) 151
automatic queue selection 75
available memory 41

B

bacct 113
batch jobs 4
 accessing files 101
 allocating processors 103
 changing execution order, *see* btop,
 bbot
 checking output, *see* bpeek
 checkpointing 113, 169
 configuration files, `lsb.params` 85
 displaying status, *see* bjobs
 e-mail about jobs 90, 113
 environment, *see* environment
 variables

exclusive 63, 67, 112
execution history, *see* bhist
input and output, *see* bsub, bpeek
interactive 145
inter-job dependencies 98
introduction 55
killing, *see* bkill
migration, *see* bmig
min/max processors 104
moving to other queues, *See*
 bswitch
pending and suspended 56, 120
pre-execution commands 97
projects 113
rerunnable 113
rerunning and restarting 174
resource requirements 91
resource reservation 50
resource usage limits 95
restarting from checkpoint, *see*
 brestart
scheduling 64
selecting hosts 93
signalling, *see* bkill, bstop,
 bresume
start and termination time 103
submitting, *see* bsub
batch queues 67
batch queues, *see* queues
batch server hosts
 displaying, *see* bhosts
bbot 131
bchkpnt 170
bclusters 189
bg, *see* job control
bhist 123, 191

bhosts 20, 79, 127, 191
 bjobs..... 22, 119, 190
 bkill..... 128
 bmgrouop 81
 bmig..... 173
 bmod..... 132
 boolean resources..... 43
 bparams 85
 bpeek..... 88, 126
 bqueues 21, 67, 127, 190
 brestart 171
 bresume 128
 bstop..... 128
 bsub..... 20, 169, 190
 input and output 90
 bswitch 131
 btop..... 131
 bugrouop 81
 BUILTIN, *see* load indices
 busers 78
 busy host status 39

C

C programming library 6
 checkpoint directory 169
 checkpoint library, *see* libckpt.a
 checkpoint period, *see* checkpointing,
 submitting jobs
 checkpointing 113
 background 166
 echckpt 167
 erestart 168
 frame arrays..... 211
 limitations..... 178
 linking programs, *see* ckpt_ld
 manual, *see* bchkpnt
 restarting jobs, *see* brestart
 submitting jobs 169
 with Alias | Wavefront software . 211

 with Fluent software 215
 chsh 150
 chunks (frame arrays)..... 203
 ckpt_crt0.o..... 175
 ckpt_ld..... 175
 ckpt_ld_f 175
 clusters 4
 commands, pre-execution, *see* pre-
 execution commands
 Condor system..... 175
 connect..... 154
 contacting Platform Computing xv
 CPU factor 27, 42
 CPU time limit..... 77
 CPU utilisation 40
 cpuf static resource 27, 42

D

daemons
 LIM (Load Information Manager) . 6
 RES (Remote Execution Server) ... 6
 dedicated resources 45
 disks
 available space, *see* tmp load index,
 maxtmp static resource
 I/O rate, *see* io load index
 number installed, *see* ndisks static
 resource
 dispatch windows..... 66
 displaying hosts, *see* bhosts, lshosts
 documentation.....xiv
 DONE batch job state 56
 DYNAMIC, *see* load indices

E

echckpt..... 167
 effective run queue length..... 40

electronic mail, *see* batch jobs, e-mail
 about jobs
 ELIM (External Load Information
 Manager) 37
 environment variables
 LS_JOBPID 179, 180, 181
 LS_SUBCWD 180
 LSB_CHKPNT_DIR 179, 215
 LSB_DEFAULTQUEUE 76
 LSB_EXIT_PRE_ABORT 180
 LSB_EXIT_REQUEUE 180
 LSB_HOSTS 103, 179
 LSB_INTERACTIVE 180
 LSB_JOBFILENAME 179
 LSB_JOBID 103, 179
 LSB_JOBINDEX 181
 LSB_JOBNAME 180
 LSB_QUEUE 180
 LSB_RESTART 180
 LSB_RESTART_PGID 169
 LSB_RESTART_PID 169
 LSF_ECHKPNTDIR 167
 LSF_JOB_STARTER 144, 180
 MP_EUIDEVICE 185
 MP_EUILIB 185
 erestart 168
 /etc/shells file 150
 exclusive jobs 112
 exclusive jobs, *see* batch jobs, exclusive
 execution priority 42
 EXIT batch job state 57
 external load index 37

F

fax numbers (Platform) xv
 fg, *see* job control
 file access 139
 files, accessing from batch jobs, *see* batch
 jobs, accessing files

finger 155
 Fluent software 215
 FORTRAN 176
 frame arrays 203

G

gbjobs 204
 gbkil 204
 gbresume 204
 gbstop 204
 gbsub 204
 guides xiv

H

help xiv, xv
 hname static resources 42
 home directory 102
 host groups 81
 host information, displaying, *see*
 lshosts
 host models 27
 displaying, *see* lsinfo
 host redirection 151
 host selection, *see* resource requirements
 host status 17, 38
 busy 39
 lockU 39, 63, 112
 lockW 39
 ok 39
 unavail 39
 unlicensed 39
 host types 42
 displaying, *see* lsinfo
 hosts file, PVM 183

I

idle time 40
 integration
 Alias | Wavefront software 211
 Fluent software 215
 interactive jobs 4
 resource reservation 51
 INTERVAL, *see* load indices
 io load index 41
 it load index 40

J

job arrays 106
 job control 152
 job dependencies 98
 job ladder, *see* job dependencies
 job migration 174
 automatic migration 174
 migration threshold 174
 job queues, *see* queues
 job spanning, *see* resource requirements
 job starter 144

L

libckpt.a 175
 LIM (Load Information Manager) 6
 load average 39
 load displaying, *see* lsload
 load index
 io 41
 it 40
 ls 40
 mem 41
 pg 40
 r15m 39
 r15s 39

 r1m 39
 swp 41
 tmp 40
 ut 40
 load indices
 built-in 27, 37
 dynamic 27
 external 37
 update interval 26
 load monitoring 29
 load sharing cluster, *see* clusters
 load sharing server 6
 load thresholds 29
 loadSched, *see* scheduling thresholds
 local mode in lstdsh 155
 locality, *see* resource requirements
 lockU host status 39, 63, 112
 lockW host status 39
 login sessions 40
 login shell 150
 ls load index 40
 LS_JOBPID environment variable. . 179,
 180, 181
 LS_SUBCWD environment variable. . 180
 LSB_CHKPT_DIR environment
 variable 179, 215
 LSB_DEFAULTQUEUE environment
 variable 76
 LSB_EXIT_PRE_ABORT environment
 variable 180
 LSB_EXIT_REQUEUE environment
 variable 180
 LSB_HOSTS environment variable. . 103,
 179
 LSB_INTERACTIVE environment
 variable 180
 LSB_JOBFILENAME environment
 variable 179
 LSB_JOBID environment variable. . 103,
 179

LSB_JOBINDEX environment variable . .
 181
 LSB_JOBNAME environment variable 180
 LSB_QUEUE environment variable . 180
 LSB_RESTART environment variable 180
 LSB_RESTART_PGID environment
 variable 169
 LSB_RESTART_PID environment
 variable 169
 LSLIB (LSF Batch LIBrary) 11
 lsclusters 188
 LSF administrator 4
 LSF base system 6
 LSF Enterprise Edition xiv
 LSF Make 19
 LSF master server, *see* master LIM
 LSF Standard Edition xiv
 LSF Suite documentation xiv
 LSF Suite products xiii
 lsf.task file 53
 LSF_ECHKPNTDIR environment
 variable 167
 LSF_JOB_STARTER environment
 variable 144, 180
 LSF_SERVERDIR directory 167
 .lsfhosts file 86, 192
 lsftask 53
 lsgrun 141
 lshosts 16, 27, 188
 lsid 25
 lsinfo 14, 26, 37, 41, 46
 LSLIB (Load Sharing LIBrary) 6
 lsload 17, 29, 41, 188
 -E option 40
 -N option 40
 lslogin 142, 192
 lsmake 159, 182
 min/max processors 163
 running as a batch job 162
 lsmode 153

lsmon 30, 188
 lsplace 142
 lsrcp 102, 140
 lsrtasks 53
 lsrun 18, 140, 192
 lstcsh 19, 149
 local and remote mode 155

M

mailing address (Platform) xv
 manager, *see* LSF administrator
 master LIM 6
 MAX_JOB_ARRAY_SIZE (in frame
 arrays) 204
 maxmem static resource 27, 42
 maxswp static resource 27, 42
 maxtmp static resource 28, 42
 mbatchd 55
 mem load index 41
 memory
 physical, *see* mem load index, maxmem
 static resource
 swap space, *see* swp load index,
 maxswp static resource
 migration, *see* bmig
 model static resource 27, 42
 monitoring load 29
 MP_EUIDEVICE environment variable .
 185
 MP_EUILIB environment variable . . 185
 MPI (Message Passing Interface) 104, 183
 MPICH 183
 mpijob 104, 184
 MPL (Message Passing Library) 183
 multiprocessor hosts
 also see ncpus static resource

N

name spaces, non-uniform 86
 ncpus static resource..... 27, 42
 ndisks static resource 28, 42
 NFS (Network File System) 139
 non-uniform name spaces..... 86
 normalized run queue length..... 40
 NQS (Network Queueing System)... 195
 DESTINATION QUEUES 196
 server..... 195

O

ok host status 39
 online documentation xv
 order string, *see* resource requirements

P

P4 104, 183
 p4job..... 104, 183
 paging rate..... 40
 parallel jobs 181
 locality..... 51
 also see lsmake
 parallel programming
 Argonne National Laboratory... 183
 Mississippi State University 183
 Oak Ridge National Laboratories 183
 P4..... 183
 PVM..... 183
 PATH environment variable 149
 PEND batch job state..... 56
 pending jobs 120
 pg load index 40
 phone numbers (Platform) xv
 Platform Computing Corporation... xv
 POE (Parallel Operating Environment) .

185

poejob..... 185
 preemptive and preemptable scheduling
 62
 pre-execution commands..... 97
 process migration, *see* bmig
 PSUSP batch job state 57, 129
 PVM..... 104, 183
 pvmjob..... 104, 183

Q

queues..... 4
 batch queues..... 67
 choosing..... 77
 displaying, *see* bqueues
 exclusive scheduling 63
 parameters
 RES_REQ 65
 RESUME_COND..... 66
 STOP_COND 65
 preemptable 62

R

r15m load index 39
 r15s load index 39
 r1m load index..... 39
 rcp 102
 re-initializing job environment..... 111
 login shell 112
 remote execution priority..... 28
 remote jobs
 execution priority..... 42
 running, *see* lsrund, lsgrun
 remote mode in lscsh 155
 remote task list, *see* task lists
 Renderer software..... 211
 rerunning batch jobs..... 113, 174

RES (Remote Execution Server)	6
resource limits	77
specifying for batch jobs	95
resource name aliases	47
resource requirements	
batch jobs	91
format	46
ordering hosts	46, 49
parallel job locality	47, 51
RES_REQ parameter	65
resource reservation	50
resource usage	47, 50
RESUME_COND parameter	66
selecting hosts	46, 47
specifying	54
STOP_COND parameter	65
resource usage, <i>see</i> resource requirements	
resources	3
boolean	43, 45
dedicated	45
displaying, <i>see</i> <code>lsinfo</code>	
listing for a host, <i>see</i> <code>lshosts</code>	
static	42
resuming jobs, <i>see</i> <code>bresume</code>	
rexpri static resource	28, 42
rlogin	142
<i>also see</i> <code>lslogin</code>	
RUN batch job state	56, 129
run queue	
effective	40
normalized	40
run windows	29, 66
rusage, <i>see</i> resource requirements	
S	
sbatchd	55
scheduling	
batch jobs	64
host partition	60
migration of rerunnable jobs	173
preemptive scheduling	62
scheduling thresholds	64
selection string, <i>see</i> resource requirements	
server shell	154
server static resource	28, 42
shell script	
mpijob	184
p4job	183
poejob	185
pvmjob	183
shell selection for jobs	116
default shell	117
span, <i>see</i> resource requirements	
SSUSP batch job state	57, 129
start time for batch jobs	103
static resource	
hname	42
static resources	41, 42
cpuf	42
maxmem	42
maxswp	42
maxtmp	42
model	42
ncpus	42
ndisks	42
rexpri	42
server	42
type	42
status	38
<i>also see</i> host status	
subdirectories, <code>lsmake</code> and	161
support	xv
suspending jobs	120
<i>also see</i> <code>bstop</code>	
swp load index	41

Index

T

task lists53
 files53
tcsh.....149
technical assistance.....xv
telephone numbers (Platform)xv
termination time for batch jobs103
thresholds
 scheduling and suspending.....64
tmp load index40
type static resource.....27, 42

U

unavail host status39
unlicensed host status.....39
user groups81
users, *see* login sessions
USUSP batch job state.....57, 129
ut load index40

X

xbsub.....23
xlsbatch.....24
xlsmon17, 30
xterm.....142